syslog-ng Premium Edition 6 LTS

Administration Guide

syslog-ng PE Administration Guide
Updated - 08 February 2021, 04:37
Version - 6 LTS

Contents

Welcome to the syslog-ng Premium Edition 6 LTS 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.

Summary of contents

Chapter 1, *Introduction to syslog-ng* describes the main functionality and purpose of syslog-ng PE.

Chapter 2, *The concepts of syslog-ng* discusses the technical concepts and philosophies behind syslog-ng PE.

Chapter 3, *Installing syslog-ng* describes how to install syslog-ng PE on various UNIX-based platforms using the precompiled binaries.

Chapter 4, *The syslog-ng PE quick-start guide* provides a briefly explains how to perform the most common log collecting tasks with syslog-ng PE.

Chapter 5, *The syslog-ng PE configuration file* discusses the configuration file format and syntax in detail, and explains how to manage large-scale configurations using included files and reusable configuration snippets.

Chapter 6, *Collecting log messages — sources and source drivers* explains how to collect and receive log messages from various sources.

Chapter 7, *Sending and storing log messages — destinations and destination drivers* describes the different methods to store and forward log messages.

Chapter 8, *Routing messages: log paths, reliability, and filters* explains how to route and sort log messages, and how to use filters to select specific messages.

Chapter 9, *Global options of syslog-ng PE* lists the global options of syslog-ng PE and explains how to use them.

Chapter 10, *TLS-encrypted message transfer* shows how to secure and authenticate log transport using TLS encryption.

Chapter 12, *Reliable Log Transfer Protocol™* describes the reliable log transport that prevents message loss.

Chapter 13, *Reliability and minimizing the loss of log messages* describes how to use flow control, disk buffering, and Reliable Log Transfer Protocol™ to minimize or completely prevent the loss of log messages.

Chapter 14, *Manipulating messages* describes how to customize message format using templates and macros, how to rewrite and modify messages, and how to use regular expressions.

Chapter 15, *Parsing and segmenting structured messages* describes how to segment and process structured messages like comma-separated values.

Chapter 16, *Processing message content with a pattern database* explains how to identify and process log messages using a pattern database.

Chapter 17, *Statistics and metrics of syslog-ng* details the available statistics that syslog-ng PE collects about the processed log messages.

Chapter 18, *Multithreading and scaling in syslog-ng PE* describes how to configure syslog-ng PE to use multiple processors, and how to optimize its performance.

Chapter 19, *Troubleshooting syslog-ng* offers tips to solving problems.

Chapter 20, *Best practices and examples* gives recommendations to configure special features of syslog-ng PE.

Appendix A, *The syslog-ng manual pages* contains the manual pages of the syslog-ng PE application.

Appendix C, *Open source licenses* includes the text of the licenses applicable to syslog-ng Premium Edition.

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) and the new syslog (IETF-syslog) protocol) standard.

Products covered in this guide

This guide describes the use of the following products:

- syslog-ng Premium Edition (syslog-ng PE) 6.0.1 and later

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 theGlossary 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.

CAUTION:

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 or menu item in the menu bar.

Button

Buttons in dialog windows.

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 syslog-ng Documentation page.

Changes in syslog-ng Premium Edition (syslog-ng PE) version 6.0.20

- Starting from syslog-ng PE version 6.0.20 and after 31 July 2020, only AIX 7 platform and syslog-ng Agent for Windows are supported. For further details about supported platforms, see Supported platforms.

Feedback

Any feedback is greatly appreciated, especially on what else this document should cover. General comments, errors found in the text, and any suggestions about how to improve the documentation is welcome at bb-pub-documentation@quest.com.

Acknowledgments

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

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

What syslog-ng is

The syslog-ng Premium Edition (syslog-ng PE) application is a flexible and highly scalable system logging application that is ideal for creating centralized and trusted logging solutions. Among others, syslog-ng PE allows you the following.

Secure and reliable log transfer

The syslog-ng PE application enables you to send the log messages of your hosts to remote servers using the latest protocol standards. You can collect and store your log data centrally on dedicated log servers. Transfer log messages using the RLTP™ protocol ensures that no messages are lost.

Disk-based message buffering. To minimize the risk of losing important log messages, the syslog-ng PE application can store messages on the local hard disk if the central log server or the network connection becomes unavailable. The syslog-ng application automatically sends the stored messages to the server when the connection is reestablished, in the same order the messages were received. The disk buffer is persistent – no messages are lost even if syslog-ng is restarted.

Secure logging using TLS. Log messages may contain sensitive information that should not be accessed by third parties. Therefore, syslog-ng PE supports the Transport Layer Security (TLS) protocol to encrypt the communication. TLS also allows you to authenticate your clients and the logserver using X.509 certificates.

Flexible data extraction and processing

Most log messages are inherently unstructured, which makes them difficult to process. To overcome this problem, syslog-ng PE comes with a set of built-in parsers, which you can combine to build very complex things.

Filter and classify. The syslog-ng PE application can sort the incoming log messages based on their content and various parameters like the source host, application, and priority. You can create directories, files, and database tables 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 PE application can segment log messages to named fields or columns, and also modify the values of these fields. You can process JSON messages, key-value pairs, and more.

To get the most information out of your log data, syslog-ng PE allows you to correlate log messages and aggregate the extracted information into a single message. You can also use external information to enrich your log data.

Wide protocol and platform support

syslog protocol standards. syslog-ng not only supports legacy BSD syslog (RFC3164) and the enhanced RFC5424 protocols, but also JavaScript Object Notation (JSON) and journald message formats.

IPv4 and IPv6 support. The syslog-ng application can operate in both IPv4 and IPv6 network environments, and can receive and send messages to both types of networks.

Client-side failover

When transferring messages to a remote server, the syslog-ng PE clients can be configured to send the log messages to secondary servers if the primary server becomes unaccessible.

Encrypted and timestamped log storage

The syslog-ng PE application can store log messages securely in encrypted, compressed, and timestamped binary files. Timestamps can be requested from an external Timestamping Authority (TSA).

Excellent performance

Depending on the exact syslog-ng PE configuration, environment, and other parameters, syslog-ng PE is capable of processing:

- Over 590,000 messages per second (over 220 MB of data per second) when receiving messages from multiple connections and storing them in text files.
- Over 560,000 messages per second (over 210 MB of data per second) when receiving

messages from multiple connections and storing them in logstore files (that is, encrypted files).

- Over 565,000 messages per second (over 210 MB of data per second) when receiving messages from multiple secure (TLS-encrypted) connections and storing them in text files.

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.

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 PE to comply with various regulations, see the Regulatory compliance and system logging whitepaper.

What is new in syslog-ng Premium Edition 6 LTS?

- For details on the news and highlights of syslog-ng Premium Edition 6 LTS, see the *Release Notes*.
- For details on changes in The syslog-ng Premium Edition 6 LTS Administrator Guide, see the section called “Version 5 F3 - 5 F4”.

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.

Public references of syslog-ng Premium Edition

Among others, the following companies decided to use syslog-ng PE in their production environment:

- Air France
- Coop Denmark
- DataPath, Inc. (Read Case Study)
- Facebook
- Hush Communications Canada Inc.
- Tecnocom Espana Solutions, S.L. (Read Case Study)

Supported platforms

Version 6.0.20 of the syslog-ng Premium Edition (syslog-ng PE) application is supported on AIX 7.1 for PowerPC.

Table of Contents

This chapter discusses the technical concepts of syslog-ng.

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, which sorts and stores them.

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 from 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.

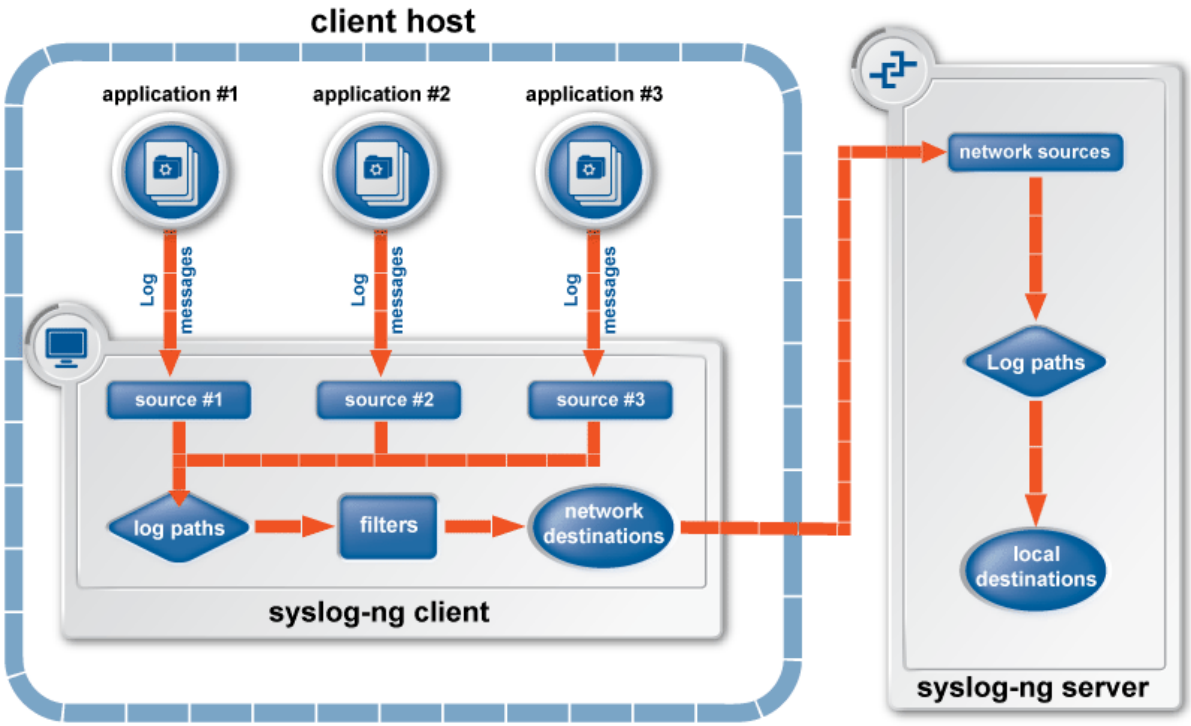
Other optional elements that can appear in log statements are *parsers* and *rewriting rules*. Parsers segment messages into different fields to help processing the messages, while rewrite rules modify the messages by adding, replacing, or removing parts of the messages.

Procedure 2.1. The route of a log message in syslog-ng

Purpose:

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.

Figure 2.1. The route of a log message



Steps:

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.
2. The syslog-ng client running on the web server reads the message from its `/var/log/apache` source.
3. The syslog-ng client processes the first log statement that includes the `/var/log/apache` source.
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.

   CAUTION:

   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. For details, see Table 8.1, "Log statement flags".

5. The syslog-ng client processes the next log statement that includes the `/var/log/apache` source, repeating Steps 3-4.
6. The message sent by the syslog-ng client arrives from a source set in the syslog-ng server.
7. The syslog-ng server reads the message from its source and processes the first log statement that includes that source.
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.

   CAUTION:

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

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

   NOTE:

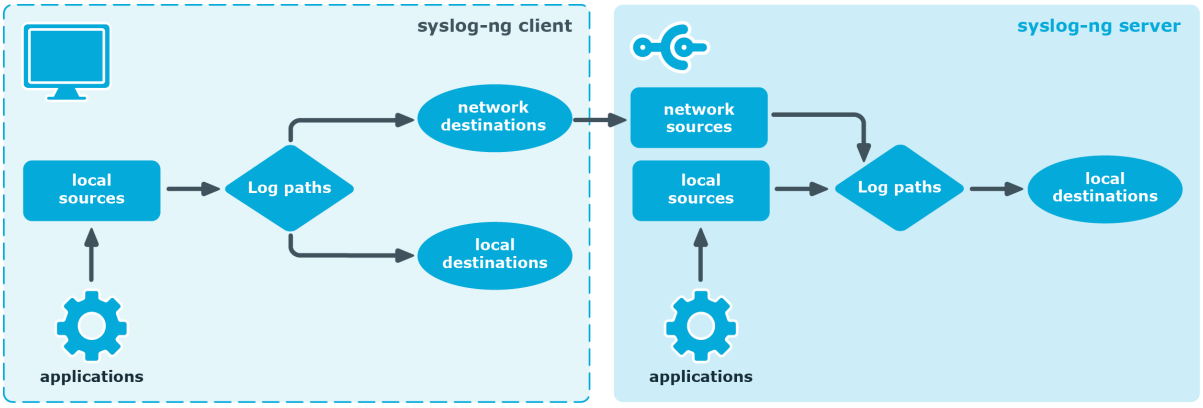
   The syslog-ng application can stop reading messages from its sources if the destinations cannot process the sent messages. This feature is called flow-control and is detailed in the section called “Managing incoming and outgoing messages with flow-control”.

Modes of operation

The syslog-ng Premium Edition application has three distinct operation scenarios: *Client*, *Server*, and *Relay*. The syslog-ng PE application running on a host determines the mode of operation automatically based on the license and the configuration file.

Client mode

Figure 2.2. Client-mode operation

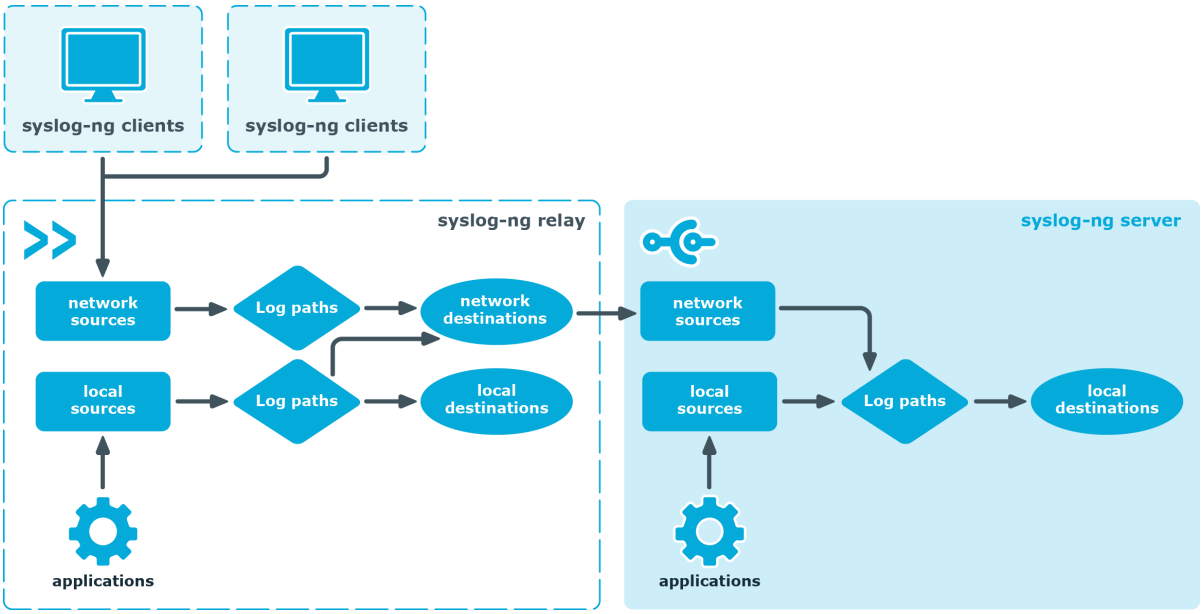


In client mode, syslog-ng collects the local logs generated by the host and forwards them through a network connection to the central syslog-ng server or to a relay. Clients often also log the messages locally into files.

No license file is required to run syslog-ng in client mode.

Relay mode

Figure 2.3. Relay-mode operation

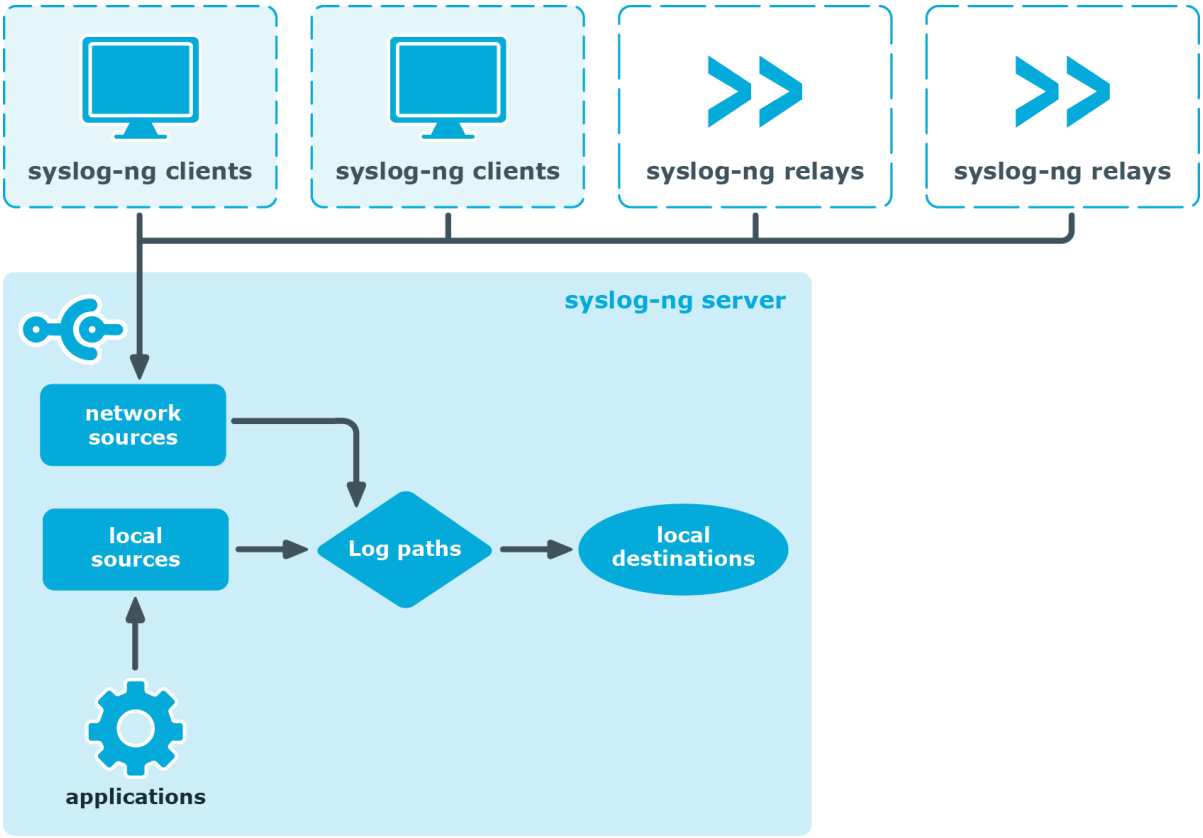


In relay mode, syslog-ng receives logs through the network from syslog-ng clients and forwards them to the central syslog-ng server using a network connection. Relays also log the messages from the relay host into a local file, or forward these messages to the central syslog-ng server.
You cannot use the following destinations in relay mode: `mongodb()`, `pipe()`, `sql()`. The `file()` and `logstore()` destinations work only for local messages that are generated on the relay.

No license file is required to run syslog-ng in relay mode.

Server mode

Figure 2.4. 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, for example log analyzers.
Running syslog-ng Premium Edition in server mode requires a license file. The license determines how many individual hosts can connect to the server. For details on how syslog-ng PE calculates the number of hosts, see the section called "Licensing".

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. For details on source drivers, see Chapter 6, *Collecting log messages — sources and source drivers*.
- *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. For details on destination drivers, see Chapter 7, *Sending and storing log messages — destinations and destination drivers*.
- *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. For details, see the section called "Customizing message format".
- *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. For details, see the section called "Customizing message format".
- *Parser*: Parsers are objects that parse the incoming messages, or parts of a message. For example, the `csv-parser()` can segment 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. For details, see Chapter 15, *Parsing and segmenting structured messages* and Chapter 16, *Processing message content with a pattern database*.
- *Rewrite rule*: A rule modifies a part of the message, for example, replaces a string, or sets a field to a specified value. For details, see the section called "Modifying messages".
- *Log paths*: A combination of sources, destinations, and other objects like filters, parsers, and rewrite rules. The syslog-ng application sends messages arriving from 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 and junctions to create complex log paths. For details, see Chapter 8, *Routing messages: log paths, reliability, and filters*.

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. For details, see the section called "Customizing message format".

- *Option*: Options set global parameters of syslog-ng, like the parameters of name resolution and timezone handling. For details, see Chapter 9, *Global options of syslog-ng PE*.

For details on the above objects, see the section called "The configuration syntax in detail".

Timezones and daylight saving

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.

The syslog-ng application supports messages originating from different timezones. The original syslog protocol (RFC3164) 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.

Procedure 2.2. How syslog-ng PE assigns timezone to the message

When syslog-ng PE receives a message, it assigns timezone information to the message using the following algorithm.

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.
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. It is not possible to override only the timezone information of the incoming message, but setting the *keep-timestamp()* option to **no** allows syslog-ng PE to replace the full timestamp (timezone included) with the time the message was received.

   NOTE:

   When processing a message that does not contain timezone information, the syslog-ng PE application will use the timezone and daylight-saving that was effective when the timestamp was generated. For example, the current time is **2011-03-11** (March 11, 2011) in the **EU/Budapest** timezone. When daylight-saving is active (summertime), the offset is **+02:00**. When daylight-saving is inactive (wintertime) the timezone offset is **+01:00**. If the timestamp of an incoming message is **2011-01-01**, the timezone associated with the message will be **+01:00**, but the timestamp will be converted, because 2011-01-01 meant winter time when daylight saving is not active but the current timezone is **+02:00**.
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.

CAUTION:

If syslog-ng PE sends the message is to the destination using the legacy-syslog protocol (RFC3164) which does not support timezone information in its timestamps, the timezone information cannot be encapsulated into the sent timestamp, so syslog-ng PE will convert the hour:min values based on the explicitly specified timezone.

4. If the timezone is not specified, local timezone is used.
5. When macro expansions are used in the destination filenames, the local timezone is used. (Also, if the timestamp of the received message does not contain the year of the message, syslog-ng PE uses the local year.)

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.)

Versions and releases of syslog-ng PE

For detailed information about syslog-ng Premium Edition (syslog-ng PE) versions and releases, visit the Product Life Cycle table on the Product Support site for syslog-ng PE.

Licensing

Licensing benefits

Buying a syslog-ng Premium Edition (syslog-ng PE) license permits you to perform the following:

- Install one instance of the syslog-ng PE application in server mode to a single host. This host acts as the central log server of the network. You have to install the license file only on this host.
- Install the syslog-ng PE application in relay or client mode on host computers within your organization (on any supported platform). You cannot redistribute the application to third parties. The total number of hosts permitted to run syslog-ng in relay or client mode is limited by the syslog-ng PE license. The client and relay hosts may use any operating system supported by syslog-ng PE. For details, see the Supported platforms in syslog-ng Premium Edition page.

The syslog-ng Premium Edition license determines the number of individual hosts (also called log source hosts) that can send log messages to syslog-ng PE.

License grants and legal restrictions are fully described in the Software Transaction, License and End User License Agreements. Note that the EULA and the syslog-ng Premium Edition Product Guide apply only to scenarios where the Licensee (the organization who has purchased the product) is the end user of the product. In any other scenario — for example, if you want to offer services provided by syslog-ng Premium Edition to your customers in an OEM or a Managed Service Provider (MSP) scenario — you have to negotiate the exact terms and conditions with One Identity.

Licensing model and modes of operation

A Log Source Host (LSH) is any host, server, or device (including virtual machines, active or passive networking devices, syslog-ng clients and relays, and so on) that is capable of sending log messages. Log Source Hosts are identified by their IP addresses, so virtual machines and vhosts are separately counted.

The syslog-ng Premium Edition application has three distinct modes of operation: Client, Relay, and Server.

- In Client mode syslog-ng Premium Edition collects local logs generated by the host it is running on, and forwards them through a network connection to the central syslog-ng PE server, a relay, or another network destination. If you install the syslog-ng Premium Edition application in Client mode on a host, it counts as a Log Source Host, even if it does not send log messages to a syslog-ng Premium Edition server.

- In Relay mode syslog-ng Premium Edition receives logs through the network from Log Source Hosts and forwards them to the central syslog-ng PE server, a relay, or another network destination. If you install the syslog-ng Premium Edition application in Relay mode on a host, it counts as a Log Source Host, even if it does not send log messages to a syslog-ng Premium Edition server.

  Relays cannot store the received log messages in local files, except for the log messages of the relay host. Naturally, relays can use disk-based buffering for every message.
- In Server mode syslog-ng Premium Edition acts as a central log-collecting server that receives messages through a network connection, and stores them locally, or forwards them to other destinations or external systems (for example, a SIEM or a database). Installing the syslog-ng Premium Edition application in Server mode requires a license file, this license file determines the number of Log Source Hosts that can send log messages to the syslog-ng Premium Edition server.

  Note that the number of source hosts is important, not the number of hosts that directly sends messages to syslog-ng Premium Edition: every host that send messages to the server (directly or using a relay) counts as a Log Source Host.

Table 2.1. Modes of operation in syslog-ng PE

	Client mode	Relay mode	Server mode
Collect the local logs of the host	✔	✔	✔
Forward local logs over the network	✔	✔	✔
Store local messages in local files	✔	✔	✔
Receive logs over the network	no	✔	✔
Forward received logs over the network	no	✔	✔
Store received logs in local files	no	no	✔
Forward logs using special destinations (for example, databases)	no	no	✔
Requires license file	no	no	✔

Notes about counting the licensed hosts

CAUTION:

- **If the actual IP address of the host differs from the IP address received by looking up its IP address from its hostname in the DNS, the syslog-ng server counts them as two different hosts.**
- **The `chain-hostnames()` option of syslog-ng can interfere with the way syslog-ng PE counts the log source hosts, causing syslog-ng to think there are more hosts logging to the central server, especially if the clients sends a hostname in the message that is different from its real hostname (as resolved from DNS). Disable the `chain-hostnames()`**

option on your log source hosts to avoid any problems related to license counting.

- **If the number of Log Source Hosts reaches the license limit, the syslog-ng PE server will not accept connections from additional hosts. The messages sent by additional hosts will be dropped, even if the client uses a reliable transport method (for example, RLTP).**
- **If the *no-parse* flag is set in a message source on the syslog-ng PE server, syslog-ng PE assumes that the message arrived from the host (that is, from the last hop) that sent the message to syslog-ng PE, and information about the original sender is lost.**

Licensing examples

Example 2.1. A simple example

Scenario:

- You want to install syslog-ng PE in server mode on a log server.
- 45 servers with syslog-ng PE installed in client mode send logs to the syslog-ng PE log server.
- 45 networks devices without syslog-ng PE installed send logs to the syslog-ng PE log server.

License requirements: You need a syslog-ng Premium Edition license for at least 100 Log Source Host (LSH) as there are 90 LSHs (45+45=90) in this scenario.

Example 2.2. High Availability (HA) cluster

Scenario:

- You want to install syslog-ng PE in server mode on two hosts that run as an active-passive high-availability cluster.
- 45 servers with syslog-ng PE installed in client mode send logs to the syslog-ng PE log server.
- 45 networks devices without syslog-ng PE installed send logs to the syslog-ng PE log server.

License requirements: You need a syslog-ng Premium Edition license for at least 100 Log Source Host (LSH) as there are 90 LSHs (45+45=90) in this scenario. You also need a High Availability (HA) license for the passive log server.

Example 2.3. Using alternative log servers with syslog-ng PE clients

Scenario:

- You want to install syslog-ng PE in server mode on a log server.
- 45 servers with syslog-ng PE installed in client mode send logs to the syslog-ng PE log server.
- 45 networks devices without syslog-ng PE installed send logs to the syslog-ng PE log server.
- 100 servers with syslog-ng PE installed send log messages to a log server without syslog-ng PE installed.

License requirements: You need a syslog-ng Premium Edition license for at least 200 LSHs as there are 190 LSHs (45+45 that send logs to a syslog-ng PE log server, and another 100 that run syslog-ng PE, 45+45+100=190) in this scenario.

Example 2.4. Using syslog-ng PE relays

Scenario:

- You want to install syslog-ng PE in server mode on a log server.
- 45 servers with syslog-ng PE installed in client mode send logs directly to the syslog-ng PE log server.
- 5 servers with syslog-ng PE installed in relay mode send logs to the syslog-ng PE log server.
- Every syslog-ng PE relay receives logs from 9 networks devices without syslog-ng PE installed (a total of 45 devices).
- 100 servers with syslog-ng PE installed send log messages to a log server without syslog-ng PE installed.

License requirements: You need syslog-ng Premium Edition license for at least 200 LSH as there are 195 LSHs (45+5+(5*9)+100=195) in this scenario.

Example 2.5. Multiple facilities

You have two facilities (for example data centers or server farms). Facility 1 has 75 AIX servers and 20 Microsoft Windows hosts, Facility 2 has 5 HP-UX servers and 40 Debian servers. That is 140 hosts altogether.

NOTE:

If, for example, the 40 Debian servers at Facility 2 are each running 3 virtual hosts, then the total number of hosts at Facility 2 is 125, and the license sizes in the following examples should be calculated accordingly.

- **Scenario:** The log messages are collected to a single, central syslog-ng PE log server.

  License requirements: You need a syslog-ng Premium Edition license for 150 LSH as there are 140 LSHs (75+20+5+40) in this scenario.

- **Scenario:** Each facility has its own syslog-ng PE log server, and there is no central log server.

  License requirements: You need two separate licenses: a license for at least 95 LSHs (75+20) at Facility 1, and a license for at least 45 LSHs (5+40) at Facility 2. You need a license for 100 LSHs at Facility 1, and a license for 50 LSHs at Facility 2.

- **Scenario:** The log messages are collected to a single, central syslog-ng PE log server. Facility 1 and 2 each have a syslog-ng PE relay that forwards the log messages to the central syslog-ng PE log server.

  License requirements: You need a syslog-ng Premium Edition license for 150 LSH as there are 142 LSHs (1+75+20+1+5+40) in this scenario (since the relays are also counted as an LSH).

- **Scenario:** Each facility to has its own local syslog-ng PE log server, and there is also a central syslog-ng PE log server that collects every log message independently from the two local log servers.

  License requirements: You need three separate licenses. A syslog-ng Premium Edition a license for at least 95 LSHs (75+20) at Facility 1, a license for at least 45 LSHs (5+40) at Facility 2, and also a license for at least 147 LSHs for the central syslog-ng Premium Edition log server (assuming that you want to collect the logs of the local log servers as well).

GPL and LGPL licenses

Starting with version 4 F1, the syslog-ng Premium Edition application is based on the syslog-ng Open Source Edition application, and includes elements that are licensed under the LGPL or GPL licenses. You can download the core of syslog-ng PE here. The components located under the `/lib` directory are licensed under the GNU Lesser General Public License Version 2.1 license, while the rest of the codebase is licensed under the GNU General Public License Version 2 license. External libraries and other dependencies used by syslog-ng PE have their own licenses, typically GPL, LGPL, MIT, or BSD.

Appendix C, *Open source licenses* includes the text of the licenses applicable to syslog-ng Premium Edition.

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 (for details, see this page).

One Identity also has a log server appliance called syslog-ng Store Box that supports high-availability. For details, see the syslog-ng Store Box Product Page.

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 the section called "BSD-syslog or legacy-syslog messages"
- The new standard described in RFC 5424 (also called the IETF-syslog protocol): see the section called "IETF-syslog messages"
- How messages are represented in syslog-ng PE: see the section called "Message representation in syslog-ng PE".

BSD-syslog or legacy-syslog messages

This section describes the format of a syslog message, according to the legacy-syslog or BSD-syslog protocol. 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 the section called "Global options". However, it is not recommended to enable messages larger than the packet size when using UDP destinations.

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.

Table 2.2. syslog Message Facilities

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)

The following table lists the severity values.

Table 2.3. syslog Message Severities

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

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 the section called "Global options".

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*.

IETF-syslog messages

This section describes the format of a syslog message, according to the IETF-syslog protocol. 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*[2]. 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.

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.

Table 2.4. syslog Message Facilities

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)

The following table lists the severity values.

Table 2.5. syslog Message Severities

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

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 the section called "Global options".

The syslog-ng PE application will truncate the following fields:

- If *APP-NAME* is longer than 48 characters it will be truncated to 48 characters.
- If *PROC-ID* is longer than 128 characters it will be truncated to 128 characters.
- If *MSGID* is longer than 32 characters it will be truncated to 32 characters.
- If *HOSTNAME* is longer than 255 characters it will be truncated to 255 characters.

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 includes 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 (for details, see the section called “Macros of syslog-ng PE”). An example STRUCTURED-DATA block looks like:

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

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*[3]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. CRLF characters will not be removed from the message.

---

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

[2] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

[3] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Message representation in syslog-ng PE

When the syslog-ng PE application receives a message, it automatically parses the message. The syslog-ng PE application can automatically parse log messages that conform to the RFC3164 (BSD or legacy-syslog) or the RFC5424 (IETF-syslog) message formats. If syslog-ng PE cannot parse a message, it results in an error.

TIP:

In case you need to relay messages that cannot be parsed without any modifications or changes, use the `flags(no-parse)` option in the source definition, and a template containing only the `${MSG}` macro in the destination definition.

To parse non-syslog messages, for example, JSON, CSV, or other messages, you can use the built-in parsers of syslog-ng PE. For details, see Chapter 15, *Parsing and segmenting structured messages*.

A parsed syslog message has the following parts.

- **Timestamps.** Two timestamps are associated with every message: one is the timestamp contained within the message (that is, when the sender sent the message), the other is the time when syslog-ng PE has actually received the message.
- **Severity.** The severity of the message.
- **Facility.** The facility that sent the message.
- **Tags.** Custom text labels added to the message that are mainly used for filtering. None of the current message transport protocols adds tags to the log messages. Tags can be added to the log message only within syslog-ng PE. The syslog-ng PE application automatically adds the id of the source as a tag to the incoming messages. Other tags can be added to the message by the pattern database, or using the `tags()` option of the source.
- **IP address of the sender.** The IP address of the host that sent the message. Note that the IP address of the sender is a hard macro and cannot be modified within syslog-ng PE but the associated hostname can be modified, for example, using rewrite rules.
- **Hard macros.** Hard macros contain data that is directly derived from the log message, for example, the ${MONTH} macro derives its value from the timestamp. The most important consideration with hard macros is that they are read-only, meaning they cannot be modified using rewrite rules or other means.
- **Soft macros.** Soft macros (sometimes also called name-value pairs) are either built-in macros automatically generated from the log message (for example, ${HOST}), or custom user-created macros generated by using the syslog-ng pattern database or a CSV-parser. The SDATA fields of RFC5424-formatted log messages become soft macros as well. In contrast with hard macros, soft macros are writable and can be modified within syslog-ng PE, for example, using rewrite rules.

NOTE:

It is also possible to set the value of built-in soft macros using parsers, for example, to set the ${HOST} macro from the message using a column of a CSV-parser.

The data extracted from the log messages using named pattern parsers in the pattern database are also soft macros.

TIP:

For the list of hard and soft macros, see the section called “Hard vs. soft macros”.

Message size and encoding

Internally, syslog-ng PE represents every message as UTF-8. The maximal length of the log messages is limited by the *log-msg-size()* option: if a message is longer than this value, syslog-ng PE truncates the message at the location it reaches the *log-msg-size()* value, and discards the rest of the message.

When encoding is set in a source (using the *encoding()* option) and the message is longer (in bytes) than *log-msg-size()* in UTF-8 representation, syslog-ng PE splits the message at an undefined location (because the conversion between different encodings is not trivial).

Structuring macros, metadata, and other value-pairs

Available in syslog-ng PE 5.1 and later.

The syslog-ng PE application allows you to select and construct name-value pairs from any information already available about the log message, or extracted from the message itself. You can directly use this structured information, for example, in the following places:

- *format-welf()* template function
- *mongodb()* destination
- or in other destinations using the *format-json()* template function.

When using *value-pairs*, there are three ways to specify which information (that is, macros or other name-value pairs) to include in the selection.

- Select groups of macros using the *scope()* parameter, and optionally remove certain macros from the group using the *exclude()* parameter.
- List specific macros to include using the *key()* parameter.
- Define new name-value pairs to include using the *pair()* parameter.

These parameters are detailed in the section called "value-pairs()".

Specifying data types in value-pairs

By default, syslog-ng PE handles every data as strings. However, certain destinations and data formats (for example, SQL, MongoDB, JSON) support other types of data as well, for example, numbers or dates. The syslog-ng PE application allows you to specify the data type in templates (this is also called type-hinting). If the destination driver supports data types, it converts the incoming data to the specified data type. For example, this allows you to store integer numbers as numbers in MongoDB, instead of strings.

CAUTION:

Hazard of data loss! If syslog-ng PE cannot convert the data into the specified type, an error occurs, and syslog-ng PE drops the message by default. To change how syslog-ng PE handles data-conversion errors, see the section called "on-error()".

To use type-hinting, enclose the macro or template containing the data with the type: **<datatype>("<macro>")**, for example: **int("$PID")**.

Currently the *mongodb()* destination and the *format-json* template function supports data types.

Example 2.6. Using type-hinting

The following example stores the MESSAGE, PID, DATE, and PROGRAM fields of a log message in a MongoDB database. The DATE and PID parts are stored as numbers instead of strings.

```
mongodb(
       value-pairs(pair("date",  datetime("$UNIXTIME"))
                   pair("pid",  int64("$PID"))
                   pair("program",  "$PROGRAM"))
                   pair("message",  "$MESSAGE"))
                     )
 );
```

The following example formats the same fields into JSON.

```
$(format-json date=datetime("$UNIXTIME") pid=int64("$PID") program="$PROGRAM"
message="$MESSAGE")
```

The syslog-ng PE application currently supports the following data-types.

- *boolean*: Converts the data to a boolean value. Anything that begins with a **t** or **1** is converted to true, anything that begins with an **f** or **0** is converted to false.
- *datetime*: Use it only with UNIX timestamps, anything else will likely result in an error. This means that currently you can use only the *$UNIXTIME* macro for this purpose.
- *literal*: The data as a literal string, without adding any quotes or escape characters.
- *int* or *int32*: 32-bit integer.
- *int64*: 64-bit integer.
- *string*: The data as a string.

value-pairs()

Type:	parameter list of the *value-pairs()* option
Default:	`empty string`

Description: The *value-pairs()* option allows you to select specific information about a message easily using predefined macro groups. The selected information is represented as

name-value pairs and can be used formatted to JSON format, or directly used in a *mongodb()* destination.

Example 2.7. Using the *value-pairs()* option

The following example selects every available information about the log message, except for the date-related macros (**R_*** and **S_***), selects the **.SDATA.meta.sequenceId** macro, and defines a new value-pair called **MSGHDR** that contains the program name and PID of the application that sent the log message.

```
value-pairs(
      scope(nv_pairs core syslog all_macros selected_macros everything)
      exclude("R_*")
      exclude("S_*")
      key(".SDATA.meta.sequenceId")
      pair("MSGHDR" "$PROGRAM[$PID]: ")
)
```

The following example selects the same information as the previous example, but converts it into JSON format.

```
$(format-json --scope nv_pairs,core,syslog,all_macros,selected_
macros,everything \
  --exclude R_* --exclude S_* --key .SDATA.meta.sequenceId \
  --pair MSGHDR="$PROGRAM[$PID]: ")
```

NOTE:

Every macro is included in the selection only once, but redundant information may appear if multiple macros include the same information (for example, including several date-related macros in the selection).

The *value-pairs()* option has the following parameters. The parameters are evaluated in the following order:

1.
2. *scope()*
3. *exclude()*
4. *key()*
5. *pair()*

   subkeys()

exclude ()

Type:	Space-separated list of macros to remove from the selection created using the *scope()* option.
Default:	empty string

Description: This option removes the specified macros from the selection. Use it to remove unneeded macros selected using the *scope()* parameter.

For example, the following example removes the SDATA macros from the selection.

```
value-pairs(
        scope(rfc5424 selected_macros)
        exclude(".SDATA*")
    )
```

The name of the macro to remove can include wildcards **(*, ?)**. Regular expressions are not supported.

key()

Type:	Space-separated list of macros to be included in selection
Default:	empty string

Description: This option selects the specified macros. The selected macros will be included as **MACRONAME = MACROVALUE**, that is using **key("HOST")** will result in **HOST = $HOST**. You can use wildcards **(*, ?)** to select multiple macros. For example:

```
value-pairs(
        scope(rfc3164)
        key("HOST"))
```

pair()

Type:	name value pairs in **"<NAME>" "<VALUE>"** format
Default:	empty string

Description: This option defines a new name-value pair to be included in the message. The value part can include macros, templates, and template functions as well. For example:

```
value-pairs(
        scope(rfc3164)
        pair("TIME" "$HOUR:$MIN")
        pair("MSGHDR" "$PROGRAM[$PID]: "))
```

rekey()

Type:	<pattern-to-select-names>, <list of transformations>
Default:	empty string

Description: This option allows you to manipulate and modify the name of the value-pairs. You can define transformations, which are are applied to the selected name-value pairs. The first parameter of the *rekey()* option is a glob pattern that selects the name-value pairs to modify. If you omit the pattern, the transformations are applied to every key of the scope. For details on globs, see the section called “glob”.

- If *rekey()* is used within a *key()* option, the name-value pairs specified in the glob of the *key()* option are transformed.
- If *rekey()* is used outside the *key()* option, every name-value pair of the *scope()* is transformed.

The following transformations are available:

add-prefix("<my-prefix>")

Adds the specified prefix to every name. For example, **rekey(add-prefix("my-prefix."))**

replace-prefix("<prefix-to-replace>", "<new-prefix>")

Replaces a substring at the beginning of the key with another string. Only prefixes can be replaced. For example, **replace-prefix(".class", ",patterndb")** changes the beginning tag **.class** to **.patterndb**

shift("<number>")

Cuts the specified number of characters from the beginning of the name.

Example 2.8. Using the rekey() option

The following sample selects every value-pair that begins with **.cee.**, deletes this prefix by cutting 4 characters from the names, and adds a new prefix (**events.**).

```
value-pairs(
    key(".cee.*"
```

```
        rekey(
            shift(4)
            add-prefix("events.")
        )
    )
)
```

The *rekey()* option can be used with the *format-json* template-function as well, using the following syntax:

```
$(format-json --rekey .cee.* --add-prefix events.)
```

scope()

Type:	space-separated list of macro groups to include in selection
Default:	empty string

Description: This option selects predefined groups of macros. The following groups are available:

- *nv-pairs*: Every soft macro (name-value pair) associated with the message, except the ones that start with a dot (.) character. Macros starting with a dot character are generated within syslog-ng PE and are not originally part of the message, therefore are not included in this group.
- *dot-nv-pairs*: Every soft macro (name-value pair) associated with the message which starts with a dot (.) character. For example, **.classifier.rule_id** and **.sdata.***. Macros starting with a dot character are generated within syslog-ng PE and are not originally part of the message.
- *all-nv-pairs*: Include every soft macro (name-value pair). Equivalent to using both *nv-pairs* and *dot-nv-pairs*.
- *rfc3164*: The macros that correspond to the RFC3164 (legacy or BSD-syslog) message format: *$FACILITY*, *$PRIORITY*, *$HOST*, *$PROGRAM*, *$PID*, *$MESSAGE*, and *$DATE*.
- *rfc5424*: The macros that correspond to the RFC5424 (IETF-syslog) message format: *$FACILITY*, *$PRIORITY*, *$HOST*, *$PROGRAM*, *$PID*, *$MESSAGE*, *$MSGID*, *$DATE*, and the metadata from the structured-data (SDATA) part of RFC5424-formatted messages, that is, every macro that starts with **.SDATA.**.

  The *rfc5424* group does not contain any metadata about the message, only information that was present in the original message. To include the most

commonly used metadata (for example, the *$SOURCEIP* macro), use the *selected-macros* group instead.

- *all-macros*: Include every hard macro. This group is mainly useful for debugging, as it contains redundant information (for example, the date-related macros include the date-related information several times in various formats).
- *selected-macros*: Include the macros of the *rfc3164* groups, and the most commonly used metadata about the log message: the *$TAGS*, *$SOURCEIP*, and *$SEQNUM* macros.
- *sdata*: The metadata from the structured-data (SDATA) part of RFC5424-formatted messages, that is, every macro that starts with **.SDATA**.
- *everything*: Include every hard and soft macros. This group is mainly useful for debugging, as it contains redundant information (for example, the date-related macros include the date-related information several times in various formats).

For example:

```
value-pairs(
        scope(rfc3164 selected-macros))
```

subkeys ()

Type:	name value pairs to select
Default:	empty string

Description: This option selects every value-pair that has a name beginning with the specified prefix, but removes the prefix when formatting the message. For example:

```
value-pairs(
        scope(rfc3164)
        subkeys(".cef.")
        )
```

Things to consider when forwarding messages between syslog-ng PE hosts

When you send your log messages from a syslog-ng PE client through the network to a syslog-ng PE server, you can use different protocols and options. Every combination has its advantages and disadvantages. The most important thing is to use matching protocols and options, so the server handles the incoming log messages properly.

In syslog-ng PE you can change many aspects of the network communication. First of all, there is the structure of the messages itself. Currently, syslog-ng PE supports two standard syslog protocols: the BSD (RFC3164) and the syslog (RFC5424) message format.

These RFCs describe the format and the structure of the log message, and add a (lightweight) framing around the messages. You can set this framing/structure by selecting the appropriate driver in syslog-ng PE. There are two drivers you can use: the *network()* driver and the *syslog()* driver. The *syslog()* driver is for the syslog (RFC5424) protocol and the network() driver is for the BSD (RFC3164) protocol.

The *tcp()* and *udp()* drivers are now deprecated, they are essentially equivalent with the *network(transport(tcp))* and *network(transport(udp))* drivers.

In addition to selecting the driver to use, both drivers allow you to use different transport-layer protocols: TCP and UDP, and optionally also higher-level transport protocols: TLS (over TCP, and RLTP (optionally using TLS). To complicate things a bit more, you can configure the *network()* driver (corresponding to the BSD (RFC3164) protocol) to send the messages in the syslog (RFC5424) format (but without the framing used in RFC5424) using the *flag(syslog-protocol)* option.

Because some combination of drivers and options are invalid, you can use the following drivers and options as sources and as destinations:

1.
2. *syslog(transport(tcp))*
3. *syslog(transport(udp))*
4. *syslog(transport(rltp))*
5. *syslog(transport(tls))*
6. *syslog(transport(rltp(tls-required(yes)))*
7. *network(transport(tcp))*
8. *network(transport(udp))*
9. *network(transport(rltp))*
10. *network(transport(tls))*

11. `network(transport(rltp(tls-required(yes)))`
12. `network(transport(tcp) flag(syslog-protocol))`
13. `network(transport(udp) flag(syslog-protocol))`
14. `network(transport(rltp)flag(syslog-protocol))`
15. `network(transport(tls) flag(syslog-protocol))`

    `network(transport(rltp(tls-required(yes)) flag(syslog-protocol))`

If you use the same driver and options in the destination of your syslog-ng PE client and the source of your syslog-ng PE server, everything should work as expected. Unfortunately there are some other combinations, that seem to work, but result in losing parts of the messages. The following table show the combinations:

Table 2.6. Source-destination driver combinations

S \ D	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	✔	-	-	-	-	!	-	-	-	-	!	-	-	-	-
2	-	✔	-	-	-	-	!	-	-	-	-	!	-	-	-
3	-	-	✔	-	-	-	-	!	-	-	-	-	!	-	-
4	-	-	-	✔	-	-	-	-	!	-	-	-	-	!	-
5	-	-	-	-	✔	-	-	-	-	!	-	-	-	-	!
6	-	-	-	-	-	✔	-	-	-	-	✔?	-	-	-	-
7	-	✔?	-	-	-	-	✔	-	-	-	-	✔?	-	-	-
8	-	-	✔?	-	-	-	-	✔	-	-	-	-	✔?	-	-
9	-	-	-	-	-	-	-	-	✔	-	-	-	-	✔?	-
10	-	-	-	-	✔?	-	-	-	-	✔	-	-	-	-	✔?
11	!	-	-	-	-	!	-	-	-	-	✔	-	-	-	-
12	-	!	-	-	-	-	!	-	-	-	-	✔	-	-	-
13	-	-	!	-	-	-	-	!	-	-	-	-	✔	-	-
14	-	-	-	!	-	-	-	-	!	-	-	-	-	✔	-
15	-	-	-	-	!	-	-	-	-	!	-	-	-	-	✔

- - This method does not work. The logs will not get to the server.
- ✔ This method works.
- ! This method has some visible drawbacks. The logs go through, but some of the values are missing/misplaced/and so on.
- ✔? This method seems to work, but it is not recommended because this can change in a future release.

The configuration options of the syslog-ng Agent for Windows application are much more limited. The configurable elements are:

- Protocol: (Legacy BSD Syslog Protocol, Syslog Protocol, Snare Protocol)
- Use SSL option
- Use syslog-ng proprietary Reliable Log Transfer Protocol (RLTP) option

The protocol corresponds to the *network()* and *syslog()* drivers of syslog-ng PE, except that the syslog-ng PE server does not support the Snare protocol.

The Use syslog-ng proprietary Reliable Log Transfer Protocol (RLTP) option is equivalent to the *transport(rltp)* option.

The Use SSL option is the Windows equivalent of the *transport(tls)* option if RLTP is not set, and to the *transport(rltp(tls-required(yes))* otherwise.

Therefore with syslog-ng Agent the following cases are possible:

Table 2.7. Source-destination driver combinations in syslog-ng Agent for Windows

The syslog-ng Agent for Windows setting	Corresponding syslog-ng PE server setting
Protocol: Syslog Protocol	*syslog(transport(tcp))*
Protocol: Syslog Protocol, Use RLTP on	*syslog(transport(rltp))*
Protocol: Syslog Protocol, Use SSL on	*syslog(transport(tls))*
Protocol: Syslog Protocol, Use RLTP on, Use SSL on	*syslog(transport(rltp(tls-required(yes)))*
Protocol: Legacy BSD Syslog Protocol	*network(transport(tcp))*
Protocol: Legacy BSD Syslog Protocol, Use RLTP on	*network(transport(rltp))*
Protocol: Legacy BSD Syslog Protocol, Use SSL on	*network(transport(tls))*
Protocol: Legacy BSD Syslog Protocol, Use RLTP on, Use SSL on	*network(transport(rltp(tls-required(yes)))*

NFS file system for log files

Using the NFS network file system can lead to problems if NFS connection is not stable, therefore One Identity does neither recommend nor officially support such scenarios. If you can avoid it, do not store log files on NFS. If the NFS connection is stable and reliable, syslog-ng PE can read and write files on mounted NFS partitions as a normal file source or destination. Read this section carefully before using syslog-ng PE and NFS-mounted log files.

Risks

If there is any issue with the NFS connection (for example, connection loss, the NFS server stops), syslog-ng PE can stop working. These NFS issues can be related to the operating system, and can also vary depending on its patch level and kernel version. The possible effects include the following:

- syslog-ng PE freezes, does not respond, does not process logs, is unable to stop or reload, and you can stop it only using the **kill -9** command
- syslog-ng PE is not able to start, and hangs during startup
- Message loss or message duplication
- Message becomes corrupt (it is not lost, but the message or some parts of it contain garbage)
- When using the *logstore()* destination, the logstore file becomes corrupt
- On some RHEL-based systems (possibly depending on the kernel version too), NFS returns NULL characters when reading a file that another process is writing at the very same moment.

Limitations of using syslog-ng PE with NFS

- Do not use the *logstore()* destination to store files on an NFS-mounted partition
- To use wildcards in the file source if your log files are on an NFS file system, set the *force-directory-polling()* option to **yes** to detect newly created files. Note that wildcard file sources are available only in syslog-ng PE version 6.0.3 and newer versions of the 6.x branch, and are not yet available in syslog-ng PE version 7.
- Since One Identity does not officially support scenarios where you use syslog-ng PE

together with NFS, One Identity will handle support requests and bugs related to such scenarios only if you can reproduce the issue independently from NFS.

Recommendations for using NFS with syslog-ng PE

If you cannot avoid using NFS with syslog-ng PE note the following points.

- USE at least NFS v4 (or newer if available)
- USE the soft mount option (`-o soft`) to mount the partition
- USE the TCP mount option (`-o tcp`) to mount the partition
- DO NOT install syslog-ng PE on an NFS-mounted partition
- DO NOT store the runtime files (for example, the configuration or the persist file) of syslog-ng PE on an NFS-mounted partition
- DO NOT use logstore on an NFS-mounted partition, it can easily become corrupted

This chapter explains how to install syslog-ng Premium Edition on the supported platforms using the precompiled binary files.

- The syslog-ng PE application features a unified installer package with identical look on every supported Linux and UNIX platforms. The generic installer, as well as installing platform-specific (for example, RPM) is described in the following sections.
- For details on installing the syslog-ng Agent for Windows application, see *Administration Guide for syslog-ng Agent for Windows*.
- If you want to manage your syslog-ng PE hosts using Puppet, see Procedure 3.10, "Managing syslog-ng PE from Puppet".

The syslog-ng PE binaries include all required libraries and dependencies of syslog-ng PE, only the `ncurses` library is required as an external dependency (syslog-ng PE itself does not use the ncurses library, it is required only during the installation). The components are installed into the `/opt/syslog-ng` directory. It can automatically re-use existing configuration and license 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 `compact` suffix does not include SQL support. If you are installing syslog-ng PE in client or relay mode, or you do not use the *`sql()`* source or destination, use the `compact` binaries. That way no unnecessary components are installed to your system.

The syslog-ng PE application can be installed interactively following the on-screen instructions as described in the section called "Installing syslog-ng using the .run installer", and also without user interaction using the silent installation option — see the section called "Installing syslog-ng PE without user-interaction".

Prerequisites to installing syslog-ng PE

- The binary installer packages of syslog-ng Premium Edition include every required dependency for most platforms, only the `ncurses` library is required as an external dependency (syslog-ng PE itself does not use the ncurses library, it is required only during the installation).

  > NOTE:
  >
  > There are two versions of every binary release. The one with the `compact` suffix does not include SQL support. If you are installing syslog-ng PE in client or relay mode, or you do not use the *`sql()`* source or destination, use the `compact` binaries. That way no unnecessary components are installed to your system.

- For Java-based destinations (for example, Elasticsearch, Apache Kafka, HDFS), Java

must be installed on the host where you use such destinations. Typically, this is the host where you are running syslog-ng PE in server mode.

- DO NOT install syslog-ng PE on an NFS-mounted partition.
- DO NOT store the runtime files (for example, the configuration or the persist file) of syslog-ng PE on an NFS-mounted partition.

Security-enhanced Linux: grsecurity, SELinux

Security-enhanced Linux solutions such as grsecurity or SELinux can interfere with the operation of syslog-ng PE. The syslog-ng PE application supports these security enhancements as follows:

- **grsecurity**: Version syslog-ng PE 5 F2 and later can be run on hosts using grsecurity, with the following limitations: using the Oracle SQL source and destination is not supported.
- **SELinux**: Version syslog-ng PE 5 F2 and later properly supports SELinux on Red Hat Enterprise Linux 6.5 and newer platforms. The CentOS platforms corresponding to the supported RHEL versions are supported as well. For details, see Procedure 3.4, “Using syslog-ng PE on SELinux”.

Installing syslog-ng using the .run installer

CAUTION:

If you already had syslog-ng Open Source Edition (OSE) installed on the host, and are upgrading to syslog-ng Premium Edition, make sure that the *${SYSLOGNG_OPTIONS}* environmental variable does not contain a `-p <path-to-pid-file>` option. If it does, remove this option from the environmental variable, because it can prevent syslog-ng PE from stopping properly. Typically, the environmental variable is set in the files `/etc/default/syslog-ng` or `/etc/sysconfig/syslog-ng`, depending on the operating system you use.

This section describes how to install the syslog-ng PE 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 PE on clients or relays, complete Procedure 3.1, "Installing syslog-ng PE in client or relay mode".
- To install syslog-ng PE on your central log server, complete Procedure 3.2, "Installing syslog-ng PE in server mode".
- To install syslog-ng PE without any user-interaction, complete the section called "Installing syslog-ng PE without user-interaction".

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.

Procedure 3.1. Installing syslog-ng PE in client or relay mode

Purpose:

Complete the following steps to install syslog-ng Premium Edition on clients or relays. For details on the different operation modes of syslog-ng PE, see the section called "Modes of operation".

Steps:

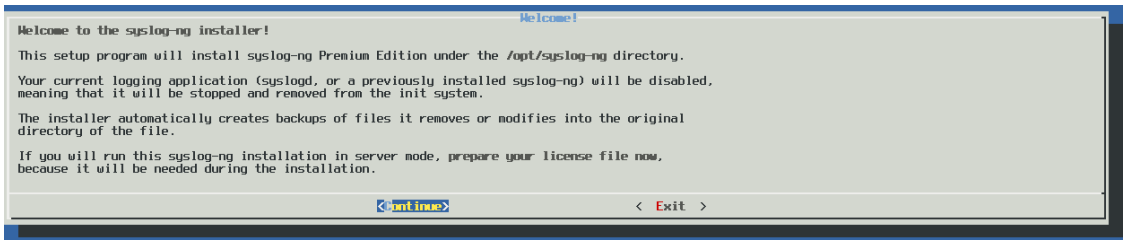
NOTE:

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.

1. Login to MyDownloads and download the syslog-ng PE installer package.

   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

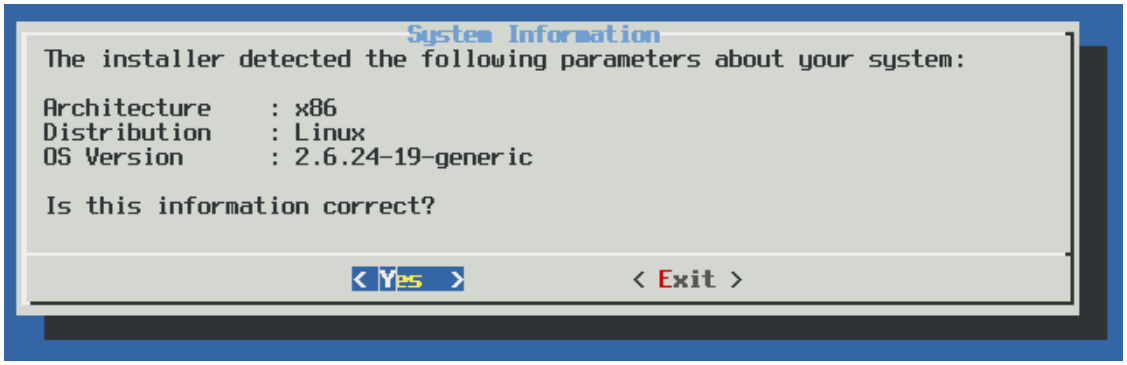
2.

3. *Accepting the EULA*: You can install syslog-ng PE 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 at Software Transaction, License and End User License Agreements. 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 PE.

   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. For a list of supported platforms, see the section called "Supported platforms". If your platform is supported but not detected correctly, contact our Support Team.

   Figure 3.2. Platform detection

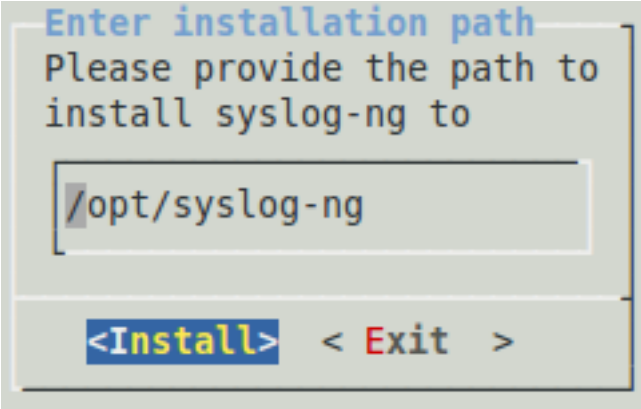
   

4.

5. *Installation path*: Enter the path to install syslog-ng PE to. This is useful if you intend

to install syslog-ng PE without registering it as a service, or if it cannot be installed to the default location because of policy compliance reasons. If no path is given, syslog-ng PE is installed to the default folder.

Figure 3.3. Installation path

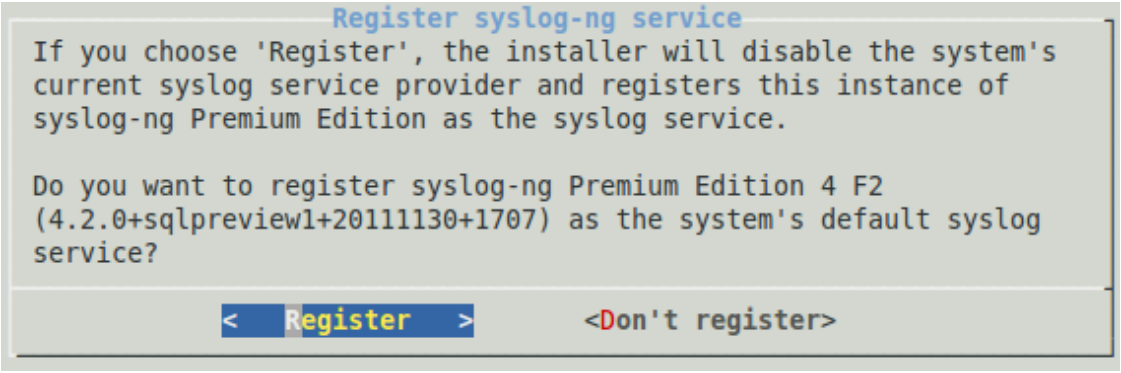


NOTE:

When installing syslog-ng PE to an alternative path on AIX, HP-UX, or Solaris platforms, set the *CHARSETALIASDIR* environmental variable to the `lib` subdirectory of the installation path. That way syslog-ng PE can find the `charset.alias` file.

Registering as syslog service: Select Register to register syslog-ng PE as the syslog service. This will stop and disable the default syslog service of the system.

Figure 3.4. Registering as syslog service



6.

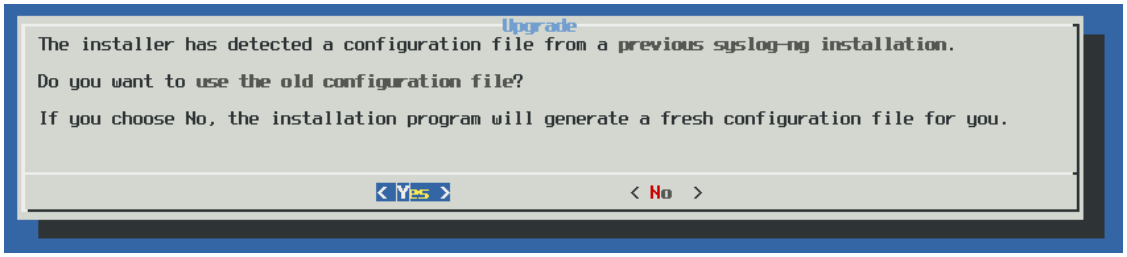
7. *Locating the license*: Since you are installing syslog-ng PE in client or relay mode, simply select OK. For details on the different operation modes of syslog-ng PE, see the section called “Modes of operation”.

8. *Upgrading*: The syslog-ng PE installer can automatically detect if you have previously installed a version of syslog-ng PE 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 3.5. Upgrading syslog-ng



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

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

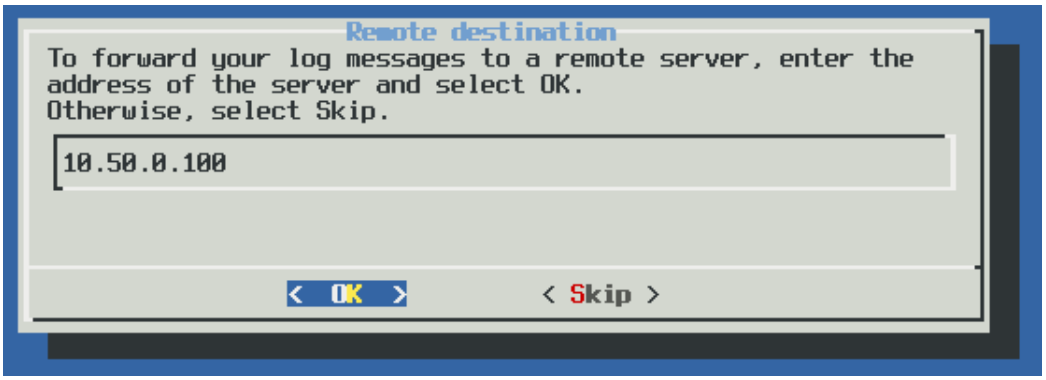
   Figure 3.6. Accepting remote messages

   a.

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

   Figure 3.7. Forwarding messages to the log server



NOTE:

Accepting remote messages and forwarding them to a log server means that syslog-ng PE will start in relay mode.

10. 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 PE 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
```

11. *Optional step for SELinux-enabled systems*: Complete Procedure 3.4, "Using syslog-ng PE on SELinux".

Procedure 3.2. Installing syslog-ng PE in server mode

Purpose:

Complete the following steps to install syslog-ng PE on log servers. For details on the different operation modes of syslog-ng PE, see the section called "Modes of operation".

Steps:

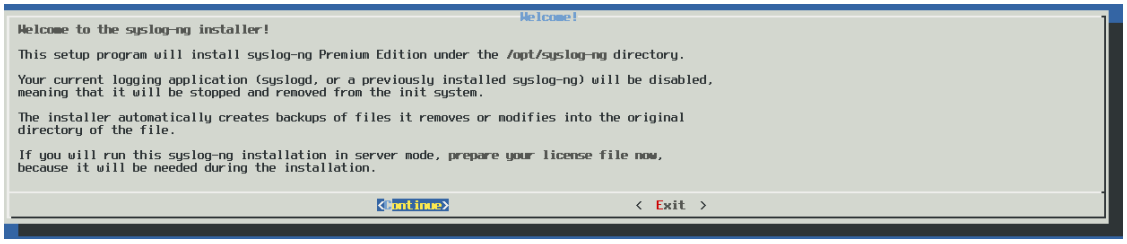
NOTE:

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.

1. Login to MyDownloads and download the syslog-ng PE installer package and your syslog-ng Premium Edition license file (*`license.txt`*). The license will be required to run syslog-ng PE in server mode (see the section called "Server mode") and is needed when you are installing syslog-ng PE on your central log server.
2. 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 3.8. The welcome screen

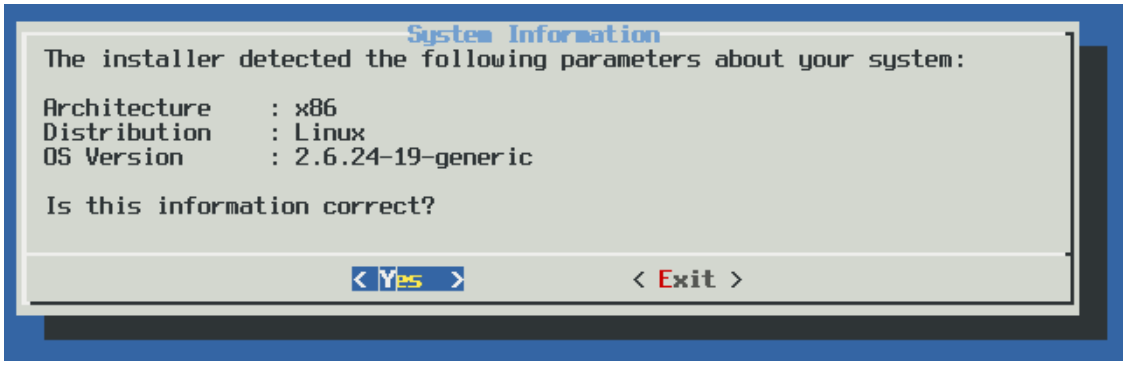


3. *Accepting the EULA*: You can install syslog-ng PE only if you understand and accept the terms of the End-User License Agreement (EULA). The full text of the EULA can be displayed during installation by selecting the Show EULA option, and is also available at Software Transaction, License and End User License Agreements. 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 PE.

   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. For a list of supported platforms, see the section called "Supported platforms". If your platform is supported but not detected correctly, contact our Support Team.

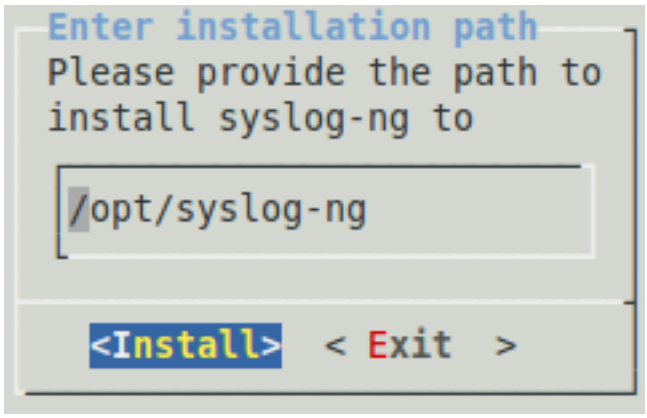
Figure 3.9. Platform detection



4.

5. *Installation path*: Enter the path to install syslog-ng PE to. This is useful if you intend to install syslog-ng PE without registering it as a service, or if it cannot be installed to the default location because of policy compliance reasons. If no path is given, syslog-ng PE is installed to the default folder.

Figure 3.10. Installation path

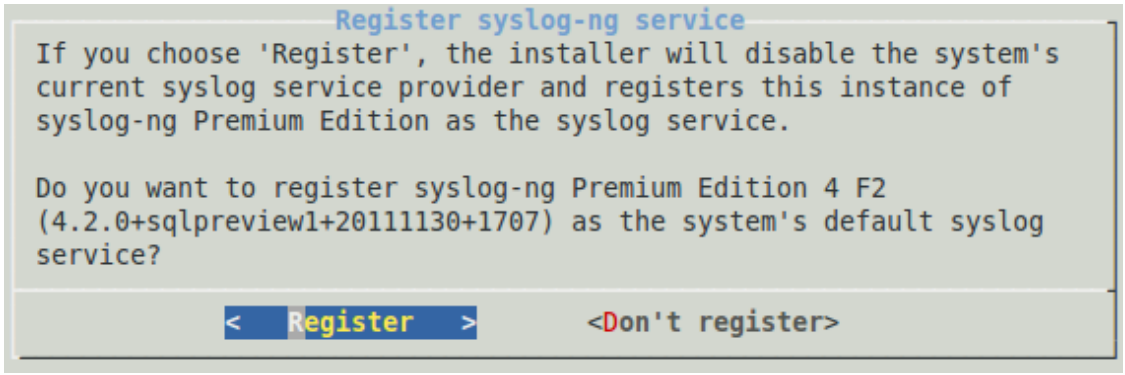


NOTE:

When installing syslog-ng PE to an alternative path on AIX, HP-UX, or Solaris platforms, set the *CHARSETALIASDIR* environmental variable to the `lib` subdirectory of the installation path. That way syslog-ng PE can find the `charset.alias` file.

Registering as syslog service: Select Register to register syslog-ng PE as the syslog service. This will stop and disable the default syslog service of the system.

Figure 3.11. Registering as syslog service

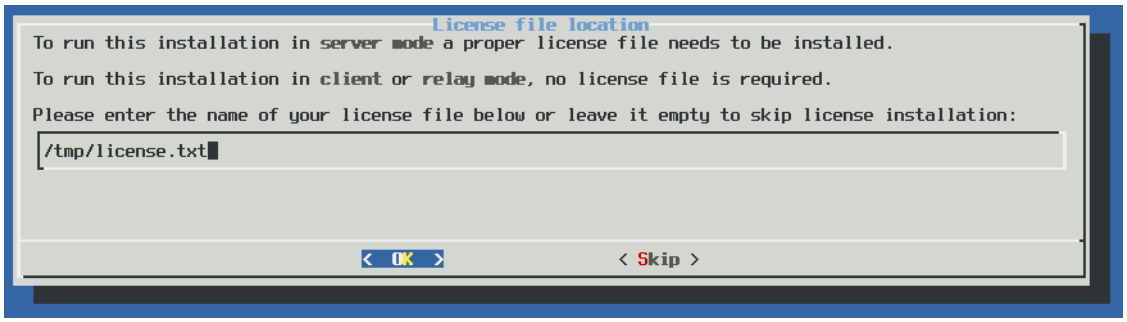


6.

7. *Locating the license*: Enter the path to your license file (`license.txt`) and select OK. Typically this is required only for your central log server.

   If you are upgrading an existing configuration that already has a license file, the installer automatically detects it.

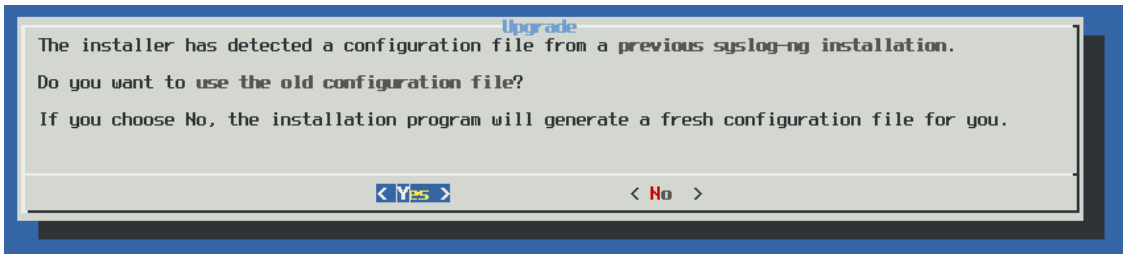
Figure 3.12. Platform detection



Upgrading: The syslog-ng PE installer can automatically detect if you have previously installed a version of syslog-ng PE 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 3.13. Upgrading syslog-ng



8.

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

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

   Figure 3.14. Accepting remote messages

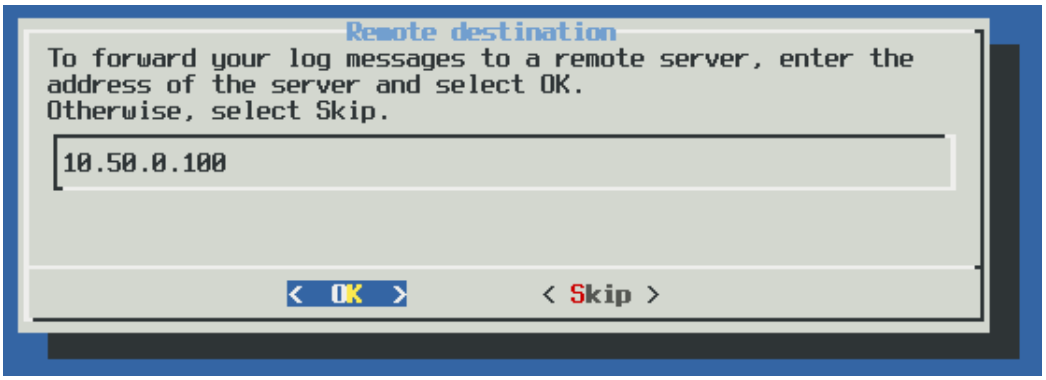
   

   a.

   b. *Remote destinations*: Enter the IP address or hostname of your log server or

relay and select OK.

Figure 3.15. Forwarding messages to the log server



> **NOTE:**
>
> Accepting remote messages and forwarding them to a log server means that syslog-ng PE will start in relay mode.

10. 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 PE 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
    ```

11. *Optional step for SELinux-enabled systems*: Complete Procedure 3.4, “Using syslog-ng PE on SELinux”.

Installing syslog-ng PE without user-interaction

The syslog-ng PE 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-premium-edition-<version>.run -- --silent [options]
```

> **CAUTION:**
>
> **The -- characters between the executable and the parameters are mandatory, like in the following example: ./syslog-ng-premium-edition-3.0.1b-solaris-10-sparc-client.run -- --silent --accept-eula -l /var/tmp/license.txt**

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

- *--accept-eula or -a*: Accept the EULA.
- *--license-file <file> or -l <file>*: Path to the license file.
- *--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.
- *--list-installed*: List information about all installed syslog-ngs.
- *--path <path>*: Set installation path.
- *--register*: Force service registration.
- *--no-register*: Prevent service registration.

Upgrading syslog-ng PE

This section describes the possible upgrade paths of syslog-ng PE.

Upgrading from syslog-ng PE to syslog-ng OSE

Upgrading from syslog-ng PE to syslog-ng OSE is unsupported since it counts as downgrading.

Upgrading from complete syslog-ng PE to client setup version of syslog-ng PE

The installer displays the following message if you try to upgrade from complete syslog-ng PE to client setup syslog-ng PE with .run package.

```
This version of syslog-ng Premium Edition doesn't support storing messages in SQL
servers, while the installed one did.
```

Uninstalling syslog-ng PE

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

- *If you have installed syslog-ng PE 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 PE, including the configuration files, use the **uninstall.sh --purge** command.
- *If you have installed syslog-ng PE from a .deb package*: Execute the **dpkg -r syslog-ng-premium-edition** command to remove syslog-ng, or the **dpkg -P syslog-ng-premium-edition** command to remove syslog-ng PE and the configuration files as well. Note that removing syslog-ng PE does not restore the syslog daemon used before syslog-ng.
- *If you have installed syslog-ng PE from an .rpm package*: Execute the **rpm -e syslog-ng-premium-edition** command to remove syslog-ng PE. Note that removing syslog-ng PE does not restore the syslog daemon used before syslog-ng PE.
- *If you have installed syslog-ng PE from a .pkg package*: Execute the **pkgrm BBsyslng** command to remove syslog-ng PE. Note that removing syslog-ng PE does not restore the syslog daemon used before syslog-ng.

  For automatic uninstall (answering **y** to all questions): Execute the **yes | pkgrm BBsyslng** command.

  The following files have to be deleted manually:

  - `<syslog-ng path>/etc/syslog-ng.conf`
  - `<syslog-ng path>/var/syslog-ng.persist`
  - `<syslog-ng path>/var/syslog-ng-00000.qf`
  - anything else under the `<syslog-ng path>/var` directory

This chapter provides a very brief introduction into configuring the syslog-ng PE application. For details on the format of the configuration file and how to configure sources, destinations, and other features, refer to the subsequent chapters.

- To configure syslog-ng PE as a client that sends log messages to a central log server, see Procedure 4.1, "Configuring syslog-ng on client hosts".
- To configure syslog-ng PE as a server that receives log messages from client hosts, see Procedure 4.2, "Configuring syslog-ng on server hosts".
- To configure syslog-ng PE as a relay that receives log messages from client hosts and forwards them to a central log server, see Procedure 4.2, "Configuring syslog-ng on server hosts".

Procedure 4.1. Configuring syslog-ng on client hosts

Purpose:

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

Steps:

1. Install the syslog-ng application on the host. For details installing syslog-ng on
2. specific operating systems, see Chapter 3, *Installing syslog-ng*.

   Configure the local sources to collect the log messages of the host. Starting with version 3.2, syslog-ng PE automatically collects the log messages that use the native system logging method of the platform, for example, messages from `/dev/log` on Linux, or `/dev/klog` on FreeBSD. For a complete list of messages that are collected automatically, see the section called "Collecting the system-specific log messages of a platform".

   Add sources to collect the messages from your log files. File sources look like this:

   ```
   source s_myfilesource {
           file("/var/log/myapplication.log" follow-freq(1)); };
   ```

   Name every source uniquely. For details on configuring file sources, see the section called "Collecting messages from text files".

   TIP:

   Many applications send log messages to logfiles by default (for example, the Roundcube webmail client, or the ProFTPD FTP server), but can be configured to send them to syslog instead. If possible, it is recommended to reconfigure the application that way.

   NOTE:

   The default configuration file of syslog-ng PE collects platform-specific log messages and the internal log messages of syslog-ng PE.

   ```
   source s_local {
   ```

```
        system();
        internal();
};
```

3. Create a network destination that points directly to the syslog-ng server, or to a local relay. The network destination greatly depends on the protocol that your log server or relay accepts messages. Many systems still use the legacy BSD-syslog protocol (RFC3162) over the unreliable UDP transport:

   ```
   destination d_network { network("10.1.2.3" transport("udp")); };
   ```

   However, if possible, use the much more reliable IETF-syslog protocol over TCP transport:

   ```
   destination d_network { syslog("10.1.2.3" transport("tcp")); };
   ```

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

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

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

   NOTE:

   The default configuration of syslog-ng PE places the collected messages into the `/var/log/messages` file:

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

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

   NOTE:

   The default configuration of syslog-ng PE has only one log statement:

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

7. Set filters, macros and other features and options (for example TLS encryption) as necessary.

Example 4.1. The default configuration file of syslog-ng PE

The following is a simple configuration file that collects local log messages to the `/var/log/messages` file.

```
@version: 6.0
@include "scl.conf"
source s_local { system(); internal(); };
destination d_local {
            file("/var/log/messages"); };
log { source(s_local); destination(d_local); };
```

Example 4.2. A simple configuration for clients

The following configuration file collects local log messages and the log messages of syslog-ng PE, and forwards them to a log server using the IETF-syslog protocol.

```
@version: 6.0
@include "scl.conf"
source s_local { system(); internal(); };
destination d_syslog_tcp {
             syslog("192.168.1.1" transport("tcp") port(2010)); };
log { source(s_local);destination(d_syslog_tcp); };
```

If you experience difficulties, see Chapter 19, *Troubleshooting syslog-ng* for tips on solving common problems.

Procedure 4.2. Configuring syslog-ng on server hosts

Purpose:

To configure syslog-ng on a server host, complete the following steps.

Steps:

1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see Chapter 3, *Installing syslog-ng*.
2. Starting with version 3.2, syslog-ng PE automatically collects the log messages that use the native system logging method of the platform, for example, messages from `/dev/log` on Linux, or `/dev/klog` on FreeBSD. For a complete list of messages that are collected automatically, see the section called "Collecting the system-specific log messages of a platform".
3. Configure the network sources that collect the log messages sent by the clients and relays. How the network sources should be configured depends also on the

capabilities of your client hosts: many older networking devices support only the legacy BSD-syslog protocol (RFC3164) using UDP transport:

```
source s_network { syslog(ip(10.1.2.3) transport("udp")); };
```

However, if possible, use the much more reliable TCP transport:

```
source s_network { syslog(ip(10.1.2.3) transport("tcp")); };
```

For other options, see the section called "Collecting messages using the IETF syslog protocol (syslog() driver)" and the section called "Collecting messages from remote hosts using the BSD syslog protocol".

> NOTE:
>
> Starting with syslog-ng PE version 3.2, the *syslog()* source driver can handle both BSD-syslog (RFC 3164) and IETF-syslog (RFC 5424-26) messages.

4. Create local destinations that will store the log messages, for example file- or program destinations. The default configuration of syslog-ng PE places the collected messages into the `/var/log/messages` file:

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

If you want to create separate logfiles for every client host, use the *${HOST}* macro when specifying the filename, for example:

```
destination d_local {
    file("/var/log/messages_${HOST}"); };
```

For details on further macros and how to use them, see Chapter 14, *Manipulating messages*.

5. Create a log statement connecting the sources to the local destinations.

```
log {
        source(s_local); source(s_network); destination(d_local); };
```

6. 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 server. This option can be set individually for every source if needed.
>
> If you are relaying log messages and want to resolve IP addresses to

hostnames, configure the first relay to do the name resolution.

Example 4.3. A simple configuration for servers

The following is a simple configuration file for syslog-ng Premium Edition that collects incoming log messages and stores them in a text file.

```
@version: 6.0
@include "scl.conf"
    options {
        time-reap(30);
        mark-freq(10);
        keep-hostname(yes);
        };
    source s_local { system(); 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); };
```

If you experience difficulties, see Chapter 19, *Troubleshooting syslog-ng* for tips on solving common problems.

Configuring syslog-ng relays

This section describes how to configure syslog-ng PE as a relay.

Procedure 4.3. Configuring syslog-ng on relay hosts

Purpose:

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

Steps:

1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see Chapter 3, *Installing syslog-ng*.
2. Configure the network sources that collect the log messages sent by the clients.
3. Create a network destination that points to the syslog-ng server.
4. Create a log statement connecting the network sources to the syslog-ng server.
5. Configure the local sources that collect the log messages of the relay host.
6. Create a log statement connecting the local sources to the syslog-ng server.
7. Enable the *keep-hostname()* and disable the *chain-hostnames()* options. (For details on how these options work, see the section called "chain-hostnames()".)

   NOTE:

   It is recommended to use these options on your syslog-ng PE server as well.
8. 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 server. This option can be set individually for every source if needed.

   If you are relaying log messages and want to resolve IP addresses to hostnames, configure the first relay to do the name resolution.

Example 4.4. 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: 6.0
@include "scl.conf"
    options {
        time-reap(30);
        mark-freq(10);
        keep-hostname(yes);
        chain-hostnames(no);
        };
    source s_local { system(); 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);
    };
```

How relaying log messages works

Depending on your exact needs about relaying log messages, there are many scenarios and syslog-ng PE options that influence how the log message will look like on the log server. Some of the most common cases are summarized in the following example.

Consider the following example: *client-host* > *syslog-ng-relay* > *syslog-ng-server*, where the IP address of *client-host* is **192.168.1.2**. The *client-host* device sends a syslog message to *syslog-ng-relay*. Depending on the settings of *syslog-ng-relay*, the following can happen.

- By default, the *keep-hostname()* option is disabled, so *syslog-ng-relay* writes the IP address of the sender host (in this case, **192.168.1.2**) to the HOST field of the syslog message, discarding any IP address or hostname that was originally in the message.
- If the *keep-hostname()* option is enabled on *syslog-ng-relay*, but name resolution is disabled (the *use-dns()* option is set to **no**), *syslog-ng-relay* uses the HOST field of the message as-is, which is probably **192.168.1.2**.
- To resolve the **192.168.1.2** IP address to a hostname on *syslog-ng-relay* using a DNS server, use the **keep-hostname(no)** and **use-dns(yes)** options. If the DNS server is properly configured and reverse DNS lookup is available for the **192.168.1.2** address, syslog-ng PE will rewrite the HOST field of the log message to *client-host*.

  NOTE:

  It is also possible to resolve IP addresses locally, without relying on the DNS server. For details on local name resolution, see Procedure 20.1, “Resolving hostnames locally”.
- The above points apply to the syslog-ng PE server (*syslog-ng-server*) as well, so if *syslog-ng-relay* is configured properly, use the **keep-hostname(yes)** option on *syslog-ng-server* to retain the proper HOST field. Setting **keep-hostname(no)** on *syslog-ng-server* would result in syslog-ng PE rewriting the HOST field to the

address of the host that sent the message to *syslog-ng-server*, which is *syslog-ng-relay* in this case.

- If you cannot or do not want to resolve the **192.168.1.2** IP address on *syslog-ng-relay*, but want to store your log messages on *syslog-ng-server* using the IP address of the original host (that is, *client-host*), you can enable the *spoof-source()* option on *syslog-ng-relay*. However, *spoof-source()* works only under the following conditions:
  - The syslog-ng PE binary has been compiled with the *--enable-spoof-source* option.
  - The log messages are sent using the highly unreliable UDP transport protocol. (Extremely unrecommended.)

Location of 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.

- On Linux and UNIX systems, the `syslog-ng.conf` and `license.txt` files are located in the `/opt/syslog-ng/etc/` directory.

NOTE:

Earlier versions of syslog-ng PE stored the configuration and license files under different directories, depending on the platform, typically under `/etc/syslog-ng/`.

NOTE:

On Microsoft Windows platforms the syslog-ng Agent for Windows stores its configuration in the system registry or in an XML file, and can be configured from a graphical interface. For details about the syslog-ng Agent for Windows, see *Administration Guide for syslog-ng Agent for Windows*.

The configuration syntax in detail

Every syslog-ng configuration file must begin with a line containing the version information of syslog-ng. For syslog-ng version 6 LTS, this line looks like:

```
@version: 6.0
```

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.

Example 5.1. 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: 6.0

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

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

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

As a syslog-ng user described on a mailing list:

The syslog-ng's config file format was written by programmers for programmers to be understood by programmers. That may not have been the stated intent, but it is how things turned out. The syntax is exactly that of C, all the way down to braces and statement terminators.

--Alan McKinnon

- The main body of the configuration file consists of object definitions: sources, destinations, log paths define which log message are received and where they are sent. 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. Object definitions (also called statements) have the following syntax:

```
object_type object_id {<options>};
```

- *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 (;).

For example, the following line defines a source and calls it **s_internal**.

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

The object can be later referenced in other statements using its ID, for example, the previous source is used as a parameter of the following log statement:

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

- The parameters and options within a statement are similar to function calls of the C programming language: the name of the option followed by a list of its parameters enclosed within brackets and terminated with a semicolon.

```
option(parameter1, parameter2); option2(parameter1, parameter2);
```

For example, the *file()* driver in the following source statement has three options: the filename (/var/log/apache/access.log), *follow-freq()*, and *flags()*. The *follow-freq()* option also has a parameter, while the *flags()* option has two parameters.

```
source s_tail { file("/var/log/apache/access.log"
    follow-freq(1) flags(no-parse, validate-utf8)); };
```

Objects may have required and optional parameters. Required parameters are positional, meaning that they must be specified in a defined order. Optional parameters 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.

Example 5.2. 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("<path-to-socket>" max-connections(10) group
(log)); };
 source s_demo_stream2 {
            unix-stream("<path-to-socket>" group(log) max-connections
(10)); };
```

- Some options are global options, or can be set globally, for example, whether syslog-ng PE should use DNS resolution to resolve IP addresses. Global options are detailed in Chapter 9, *Global options of syslog-ng PE*.

  ```
  options { use-dns(no); };
  ```

- All identifiers, attributes, and any other strings used in the syslog-ng configuration file are case sensitive.
- Objects can be used before definition.
- 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("<path-to-socket>" max-connections(10) group(log)); };
  ```

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.

Notes about the configuration syntax

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

- The configuration file can contain a maximum of 6665 source / destination / log elements.
- 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.
- Numbers 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("<path-to-socket>" max-connections(10) group(log)); };
source s_demo_stream {
        unix-stream("<path-to-socket>", max-connections(10), group(log)); };
```

- 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("<path-to-socket>" max-connections(10) group(log)); };
```

- For notes on using regular expressions, see the section called “Regular expressions”.

Global and environmental variables

Starting with syslog-ng PE version 4 F1, it is possible to define global variables in the configuration file. Global variables are actually **name-value** pairs. When syslog-ng processes the configuration file during startup, it automatically replaces **`name`** with **value**. To define a global variable, use the following syntax:

```
@define name "value"
```

The value can be any string, but special characters must be escaped. To use the variable, insert the name of the variable enclosed between backticks (`, similarly to using variables in Linux or UNIX shells) anywhere in the configuration file. If backticks are meant literally, repeat the backticks to escape them. For example, ``not-substituted-value``.

The value of the global variable can be also specified using the following methods:

- Without any quotes, as long as the value does not contain any spaces or special characters. In other word, it contains only the following characters: **a-zA-Z0-9_.**.
- Between apostrophes, in case the value does not contain apostrophes.
- Between double quotes, in which case special characters must be escaped using backslashes (\).

TIP:

The environmental variables of the host are automatically imported and can be used as global variables.

Example 5.3. Using global variables

For example, if an application is creating multiple log files in a directory, you can store the path in a global variable, and use it in your source definitions.

```
@define mypath "/opt/myapp/logs"
        source s_myapp_1 { file("`mypath`/access.log" follow-freq(1)); };
        source s_myapp_2 { file("`mypath`/error.log" follow-freq(1)); };
        source s_myapp_3 { file("`mypath`/debug.log" follow-freq(1)); };
```

The syslog-ng PE application will interpret this as:

```
@define mypath "/opt/myapp/logs"
            source s_myapp_1 { file("/opt/myapp/logs/access.log" follow-
freq(1)); };
            source s_myapp_2 { file("/opt/myapp/logs/error.log" follow-
freq(1)); };
            source s_myapp_3 { file("/opt/myapp/logs/debug.log" follow-
freq(1)); };
```

Logging configuration changes

Every time syslog-ng is started, or its configuration is reloaded, it automatically logs the SHA-1 fingerprint of its configuration file using the *internal()* message source. That way any modification of the configuration of your syslog-ng clients is visible in the central logs. Note that the log message does not contain the exact change, nor can the configuration file be retrieved from the fingerprint. Only the fact of the configuration change can be detected.

NOTE:

Modular configuration files that are included in the main configuration file of syslog-ng PE are included when the fingerprint is calculated. However, other external files (for example, scripts used in program sources or destinations) are excluded, therefore their modifications do not change the fingerprint.

The fingerprint can be examined with the **logchksign** command-line application, which detects that the fingerprint was indeed generated by a syslog-ng application. Just paste the hashes from the log message after the logchksign command like in the following example:

```
bin/logchksign "cfg-fingerprint='832ef664ff79df8afc66cd955c0c8aaa3c343f31', cfg-
nonce-ndx='0', cfg-signature='785223cfa19ad52b855550be141b00306347b0a9'"
```

Modules in syslog-ng PE

The syslog-ng PE application is modular, to increase its flexibility and also to simplify the development of additional modules. Most of the functionality of syslog-ng PE is in separate modules. That way it becomes also possible to finetune the resource requirements of syslog-ng PE, for example, by loading only the modules that are actually used in the configuration, or simply omitting modules that are not used but require large amount of memory.

Each module contains one or more plugins, which add some functionality to syslog-ng PE, for example, a destination or a source driver.

- To display the list of available modules, execute the **syslog-ng --version** command.
- To the description of the available modules, execute the **syslog-ng --module-registry** command.
- To customize which modules are loaded automatically when syslog-ng PE is started, use the **--default-modules** command-line option of syslog-ng PE.
- To request loading a module from the syslog-ng PE configuration file, see the section called “Loading modules”.

For details on the command-line parameters of syslog-ng PE mentioned in the previous list, see the syslog-ng PE man page at syslog-ng(8).

Loading modules

The syslog-ng Premium Edition application loads every available module during startup, except the *snmp()* module, and Java-related modules like *hdfs()* (Java-related modules require the *mod-java* module). For details on using the *snmp()* destination driver, see the section called “Sending SNMP traps”.

To load a module that is not loaded automatically, include the following statement in the syslog-ng PE configuration file:

```
@module <module-name>
```

Note the following points about the *@module* statement:

- The *@module* statement is a top-level statement, that is, it cannot be nested into any other statement. Usually it is used immediately after the *@version* statement.
- Every *@module* statement loads a single module: loading multiple modules requires a separate *@module* statement for every module.
- In the configuration file, the *@module* statement of a module must be earlier than the module is used.

Managing complex syslog-ng configurations

The following sections describe some methods that can be useful to simplify the management of large-scale syslog-ng PE installations. If you are using Puppet to manage your IT infrastructure, you can use it to manage your syslog-ng PE configurations as well. For details, see Procedure 3.10, “Managing syslog-ng PE from Puppet”.

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 "<path to filename>"
```

> **NOTE:**
>
> If you enter only the filename, syslog-ng PE will search for the file in the default directory: `/opt/syslog-ng/etc`. If syslog-ng PE has been installed to a different directory, use the full path instead.

This imports the entire file into the configuration of syslog-ng PE, at the location of the include statement. The <filename> can be one of the following:

- A filename, optionally with full path. The filename (not the path) can include UNIX-style wildcard characters (`*`, `?`). When using wildcard characters, syslog-ng PE will include every matching file. For details on using wildcard characters, see the section called “glob”.
- A directory. When including a directory, syslog-ng PE will try to include every file from the directory, except files beginning with a ~ (tilde) or a . (dot) character. Including a directory is not recursive. The files are included in alphabetic order, first files beginning with uppercase characters, then files beginning with lowercase characters. For example, if the directory contains the **`a.conf`**, **`B. conf`**, **`c.conf`**, **`D.conf`** files, they will be included in the following order: **`B.conf`**, **`D. conf`**, **`a.conf`**, **`c.conf`**.

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.
-

  You cannot include complete configuration files into each other, only configuration snippets can be included. This means that the included file cannot have a *@version* statement.

- Include statements can only be used at top level of the configuration file. For example, the following is correct:

  ```
  @version: 6.0
  @include "example.conf"
  ```

  But the following is not:

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

CAUTION:

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.

Reusing configuration blocks

To create a reusable configuration snippet and reuse parts of a configuration file, you have to define the block (for example, a source) once, and reference it later. (Such reusable blocks are sometimes called a Source Configuration Library, or SCL.) Any syslog-ng object can be a block. Use the following syntax to define a block:

```
block type name() {<contents of the block>};
```

Type must be one of the following: *destination*, *filter*, *log*, *parser*, *rewrite*, *root*, *source*. The *root* blocks can be used in the "root" context of the configuration file, that is, outside any other statements.

Blocks may be nested into each other, so for example a block can be built from other blocks. Blocks are somewhat similar to C++ templates.

The type and name combination of each block must be unique, that is, two blocks can have the same name if their type is different.

To use a block in your configuration file, you have to do two things:

- Include the file defining the block in the `syslog-ng.conf` file — or a file already included into `syslog-ng.conf`.
- Reference the name of the block in your configuration file. This will insert the block into your configuration. For example, to use a block called **`myblock`**, include the following line in your configuration:

  ```
  myblock()
  ```

  Blocks may have parameters, but even if they do not, the reference must include opening and closing parentheses like in the previous example.

The contents of the block will be inserted into the configuration when syslog-ng PE is started or reloaded.

Example 5.4. Reusing configuration blocks

Suppose you are running an application on your hosts that logs into the `/opt/var/myapplication.log` file. Create a file (for example, `myblocks.conf`) that stores a source describing this file and how it should be read:

```
block source myappsource() {
        file("/opt/var/myapplication.log" follow-freq(1) default-facility
(syslog)); };
```

Include this file in your main syslog-ng configuration file, reference the block, and use it in a logpath:

```
@version: 6.0
@include "<correct/path>/myblocks.conf"
source s_myappsource { myappsource(); };
...
log { source(s_myappsource); destination(...); };
```

To define a block that defines more than one object, use *`root`* as the type of the block, and reference the block from the main part of the syslog-ng PE configuration file.

Example 5.5. Defining blocks with multiple elements

The following example defines a source, a destination, and a log path to connect them.

```
block root mylogs() {
            source s_file { file("/var/log/mylogs.log" follow-freq(1)); };
            destination d_local { file("/var/log/messages"); };
            log { source(s_file); destination(d_local); };
};
```

TIP:

Since the block is inserted into the syslog-ng PE configuration when syslog-ng PE is started, the block can be generated dynamically using an external script if needed. This is useful when you are running syslog-ng PE on different hosts and you want to keep the main configuration identical.

If you want to reuse more than a single configuration object, for example, a logpath and the definitions of its sources and destinations, use the include feature to reuse the entire snippet. For details, see the section called “Including configuration files”.

Passing arguments to configuration blocks

Configuration blocks can receive arguments as well. The parameters the block can receive must be specified when the block is defined, using the following syntax:

```
block type block_name(argument1(<default-value-of-the-argument>) argument2(<default-value-of-the-argument>) argument3())
```

If an argument does not have a default value, add parentheses with empty quotes after the name of the argument, like this: `example-option("")`. To refer the value of the argument in the block, use the name of the argument between backticks (for example, **`` `argument1` ``**).

Example 5.6. Passing arguments to blocks

The following sample defines a file source block, which can receive the name of the file as a parameter. If no parameter is set, it reads messages from the `/var/log/messages` file.

```
block source s_logfile (filename("messages")) {
    file("/var/log/`filename`" );
};

source s_example {
    s_logfile(filename("logfile.log"));
};
```

If you reference the block with more arguments then specified in its definition, you can use these additional arguments as a single argument-list within the block. That way, you can use a variable number of optional arguments in your block. This can be useful when passing arguments to a template, or optional arguments to an underlying driver. To reference this argument-list, insert `__**VARARGS**__` to the place in the block where you want to insert the argument-list. Note that you can use this only once in a block. The following definition extends the logfile block from the previous example, and passes the optional arguments (**`follow-freq(1) flags(no-parse)`**) to the **`file()`** source.

```
block source s_logfile (filename("messages")) {
   file("/var/log/`filename`" `__VARARGS__`);
};

source s_example {
   s_logfile(filename("logfile.log") follow-freq(1) flags(no-parse));
};
```

How sources work

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 6.1. 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 { network(ip(10.1.2.3) port(1999)); };
```

Example 6.2. 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 {
          network(ip(10.1.2.3) port(1999));
          network(ip(10.1.2.3) port(1999) transport("udp")); };
```

Example 6.3. 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 { network(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, and so on) 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.

CAUTION:

Sources and destinations are initialized only when they are used in a log statement. For example, syslog-ng PE starts listening on a port or starts polling a file only if the source is used in a log statement. For details on creating log statements, see Chapter 8, *Routing messages: log paths, reliability, and filters*.

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:

Table 6.1. Communication methods used between the applications and syslogd

Platform	Method
Linux	A *SOCK_DGRAM* unix socket named `/dev/log`. Newer distributions that use systemd collect log messages into a journal file.
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 multi-threaded 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`.

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 6.4. Source statement on a Linux based operating system

The following source statement collects the following log messages:

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

```
source s_demo {
    internal();
    network(transport("udp"));
    unix-dgram("/dev/log"); };
```

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

Table 6.2. Source drivers available in syslog-ng

Name	Description
file()	Opens the specified file and reads messages.
internal()	Messages generated internally in syslog-ng.
network()	Receives messages from remote hosts using the BSD-syslog protocol over IPv4 and IPv6. Supports the TCP, UDP,RLTP™, and TLS network protocols.
pipe()	Opens the specified named pipe and reads messages.
program()	Opens the specified application and reads messages from its standard output.
sql()	Collects logs from tables of relational database
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.
system()	Automatically detects which platform syslog-ng PE is running on, and collects the native log messages of that platform.
systemd-journal()	Collects messages directly from the journal of platforms that use systemd.
systemd-syslog()	Collects messages from the journal using a socket on platforms that use systemd.
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.

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 6.6. Using the internal() driver

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

The syslog-ng PE application sends the following message types from the internal() source:

- *fatal*: Priority value: critical (2), Facility value: syslog (5)
- *error*: Priority value: error (3), Facility value: syslog (5)
- *warning*: Priority value: warning (4), Facility value: syslog (5)
- *notice*: Priority value: notice (5), Facility value: syslog (5)
- *info*: Priority value: info (6), Facility value: syslog (5)

internal() source options

The `internal()` driver has the following options:

host-override()

Type:	string
Default:	

Description: Replaces the ${HOST} part of the message with the parameter string.

log-iw-size()

Type:	number (messages)

Default: 1000

Description: The size of the initial window, this value is used during flow control. If the *max-connections()* option is set, the *log-iw-size()* will be divided by the number of connections, otherwise *log-iw-size()* is divided by 10 (the default value of the *max-connections()* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.7. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

normalize-hostnames()

Accepted values:	**yes** | **no**
Default:	**no**

Description: If enabled (**normalize-hostnames(yes)**), syslog-ng PE converts the hostnames to lowercase.

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`.

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.

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-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is `yes`, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

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 in the `/opt/syslog-ng/var/syslog-ng.persist` file, 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 the section called “file() source options”.

CAUTION:

Hazard of data loss! If your log files are on an NFS-mounted network file system, see the section called “NFS file system for log files”.

Declaration:

```
file("filename");
```

Example 6.8. Using the file() driver

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

Example 6.9. 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)); };
```

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.

Wildcards and file sources.

In syslog-ng PE, the filename (but not the pathname) may include wildcard characters (for example *). Note that when using wildcards in filenames, always set how often syslog-ng PE should check the file for new messages using the *follow-freq()* parameter.

CAUTION:

If you use wildcards in multiple file sources, make sure that the files and folders that match the wildcards do not overlap. That is, every file and folder should belong to only one file source. Monitoring a file from multiple wildcard sources can lead to data loss.

To use wildcards in the file source if your log files are on an NFS file system, set the *force-directory-polling()* option to **yes** to detect newly created files. Note that wildcard file sources are available only in syslog-ng PE version 6.0.3 and newer versions of the 6.x branch, and are not yet available in syslog-ng PE version 7.

When using wildcards, syslog-ng PE monitors every matching file, and can receive new log messages from any of the files. However, monitoring (polling) many files (that is, more than ten) has a significant overhead and may affect performance. On Linux this overhead is not so significant, because syslog-ng PE uses the inotify feature of the kernel.

Also, by default, the operating system notifies syslog-ng PE when an application modifies a logfile. However, in some cases this does not happen, because the file-monitoring API of the operating system does not notice that the file has changed. In such cases, enable the *force-directory-polling()* option. Note that enabling this option decreases the performance of syslog-ng PE if you monitor lots of logfiles.

Example 6.10. Using wildcards in the filename

The following example monitors every file with the `.log` extension in the `/var/application` directory for log messages. Note that only syslog-ng PE supports wildcards in the file and pathnames.

```
source s_file { file("/var/application/*.log" follow-freq(1));};
```

Notes on reading kernel messages

Note the following points when reading kernel messages on various platforms.

- 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.
- 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 and so on, 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.

- To read the kernel messages on HP-UX platforms, use the following options in the source statement:

```
file("/dev/klog" program-override("kernel") flags(kernel) follow-freq(0));
```

File sources and the RFC5424 message format

When reading messages from a file and forwarding them in IETF-syslog (RFC5424) format, syslog-ng PE automatically adds all file-related information to the **`file@18372.4`** SDATA block. When the source is file and the transport protocol is syslog or syslog-protocol flags were used in the destination side, the message will contain the following source file-related information:

- *size*: size of the file
- *position*: file position the message was read from
- *name*: name of the file

Example 6.11. File-related information in message

```
309 <38>1 2010-10-19T15:50:45.018203+02:00 server1 localprg 1234 -
[timeQuality isSynced="0" tzKnown="0"][file@18372.4 size="184567" pos="1024"
name="/var/tmp/msg.txt"] seq: 0000000001, runid: 1287496244, stamp: 2010-10-
19T15:50:45 messagetext
```

file() source options

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. For example, **`default-priority(warning)`**

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. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

flags()

Type:	assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse, dont-store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The *`assume-utf8`* flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the *`validate-utf8`* flag.
- *dont-store-legacy-msghdr*: By default, 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`**). If you do not want to store the original header of the message, enable the *`dont-store-legacy-msghdr`* flag.

- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the **LOG_KERN | LOG_NOTICE** priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```
- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the `rltp`, `syslog()`, `network()`, `unix-dgram()` drivers support multi-line messages.
- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the `flags(no-parse)` option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *syslog-protocol*: The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the `syslog` driver, which handles only messages that have a frame header.
- *validate-utf8*: The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section called “IETF-syslog messages”). If the*BOM*[4]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 (seconds)
Default:	1

Description: Indicates that the source should be checked periodically. 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.

force-directory-polling()

Type:	yes or no
Default:	no

Description: Specifies whether syslog-ng should force the polling of the logfiles that match the wildcarded filenames specified in the *file()* driver.

NOTE:

Enabling this option decreases performance if you monitor lots of logfiles.

To use wildcards in the file source if your log files are on an NFS file system, set the *force-directory-polling()* option to **yes** to detect newly created files. Note that wildcard file sources are available only in syslog-ng PE version 6.0.3 and newer versions of the 6.x branch, and are not yet available in syslog-ng PE version 7.

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.

CAUTION:

To use the `s_` macros, the *keep-timestamp()* option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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:	1000

Description: The size of the initial window, this value is used during flow control. Make sure that *log-iw-size()* is larger than the value of *log-fetch-limit()*.

When using wildcards in the filenames, syslog-ng PE attempts to read *log-fetch-limit()* number of messages from each file. For optimal performance, make sure that *log-iw-size()* is greater than **log-fetch-limit()*(the-number-of-matching-files)**.

Example 6.12. Initial window size of file sources

If **log-fetch-limit()** is 10, and your wildcard file source has 200 files, then *log-iw-size()* should be at least 2000.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

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.

multi-line-garbage()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-garbage()* option when processing multi-line messages that contain unneeded parts between the messages. Specify a string or regular expression that matches the beginning of the unneeded message parts. If the *multi-line-garbage()* option is set, syslog-ng PE ignores the lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*. See also the *multi-line-prefix()* option.

When receiving multi-line messages from a source when the *multi-line-garbage()* option is set, but no matching line is received between two lines that match *multi-line-prefix()*, syslog-ng PE will continue to process the incoming lines as a single message until a line matching *multi-line-garbage()* is received.

CAUTION:

If the *multi-line-garbage()* option is set, syslog-ng PE discards lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

multi-line-prefix()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-prefix()* option to process multi-line messages, that is, log messages that contain newline characters (for example, Tomcat logs). Specify a string or regular expression that matches the beginning of the log messages. Use as simple regular expressions as possible, because complex regular expressions can severely reduce the rate of processing multi-line messages. If the *multi-line-prefix()* option is set, syslog-ng PE ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the *multi-line-garbage()* option.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

TIP:

•

To make multi-line messages more readable when written to a file, use a template in the destination and instead of the *${MESSAGE}* macro, use the following: *$(indent-multi-line ${MESSAGE})*. This expression inserts a tab after every newline character (except when a tab is already present), indenting every line of the message after the first. For example:

```
destination d_file {
    file ("/var/log/messages"
        template("${ISODATE} ${HOST} $(indent-multi-line ${MESSAGE})\n")
);
};
```

- For details on using templates, see the section called "Templates and macros".

To actually convert the lines of multi-line messages to single line (by replacing the newline characters with whitespaces), use the *flags(no-multi-line)* option in the source.

Example 6.13. Processing Tomcat logs

The log messages of the Apache Tomcat server are a typical example for multi-line log messages. The messages start with the date and time of the query in the **YYYY.MM.DD HH:MM:SS** format, as you can see in the following example.

```
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
SEVERE: Catalina.start:
LifecycleException:  service.getName(): "Catalina";  Protocol handler start
failed: java.net.BindException: Address already in use<null>:8080
       at org.apache.catalina.connector.Connector.start(Connector.java:1138)
       at org.apache.catalina.core.StandardService.start
(StandardService.java:531)
       at org.apache.catalina.core.StandardServer.start
(StandardServer.java:710)
       at org.apache.catalina.startup.Catalina.start(Catalina.java:583)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:288)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
```

```
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.apache.commons.daemon.support.DaemonLoader.start
(DaemonLoader.java:177)
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
INFO: Server startup in 1206 ms
2010.06.09. 12:45:08 org.apache.coyote.http11.Http11Protocol pause
INFO: Pausing Coyote HTTP/1.1 on http-8080
2010.06.09. 12:45:09 org.apache.catalina.core.StandardService stop
INFO: Stopping service Catalina
```

To process these messages, specify a regular expression matching the timestamp of the messages in the *multi-line-prefix()* option. Such an expression is the following:

```
source s_file{ file("/var/log/tomcat6/catalina.2010-06-09.log" follow-freq
(0) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]{2}\.") flags(no-parse));
 };
```

Note that the **flags(no-parse)** is needed to avoid syslog-ng PE trying to interpret the date in the message.

pad-size()

Type:	number (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng PE 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. If *pad-size()* was given and the incoming message does not fit into *pad-size()*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough 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`.

read-old-records()

Type:	yes|no
Default:	yes

Description: If set to **`yes`**, syslog-ng PE will start reading the records from the beginning of the file, if the file has not been read yet. If set to **`no`**, syslog-ng PE will read only the new records. If the source has a state in the persist file, this option will have no effect.

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.

Example 6.14. Monitoring multiple directories

The following example reads files having the `.log` extension from the `/var/application/` directory and its subdirectories. Note that only syslog-ng PE supports recursive directory handling and wildcards in the file and pathnames.

```
source s_file_subdirectories { file("/var/application/*.log"
                    recursive(yes)
                    follow-freq(1)
                    log-fetch-limit(100)
                    );};
```

replace-null-characters()

Type:	yes|no
Default:	no

Description: If set to **yes**, syslog-ng PE replaces the '\0' characters in the file input data with spaces ' '. Available in version 6.0.8 and later.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the /usr/share/zoneinfo directory.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is **yes**, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[4] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Collecting messages using the RFC3164 protocol (network() driver)

The network() source driver can receive syslog messages conforming to RFC3164 from the network using the TCP, TLS, and UDP networking protocols.

You can use the RLTP™ protocol as well. For details about the RLTP™ protocol, see Chapter 12, *Reliable Log Transfer Protocol™* .

- UDP is a simple datagram oriented protocol, which provides "best effort service" to transfer messages between hosts. It may lose messages, and no attempt is made to retransmit lost messages. The *BSD-syslog* protocol traditionally uses UDP.

  Use UDP only if you have no other choice.

  For details on minimizing message loss when using UDP, see *Collecting log messages from UDP sources*.
- TCP provides connection-oriented service: the client and the server establish a connection, each message is acknowledged, and lost packets are resent. TCP can detect lost connections, and messages are lost, only if the TCP connection breaks. When a TCP connection is broken, messages that the client has sent but were not yet received on the server are lost.
- The syslog-ng application supports TLS (Transport Layer Security, also known as SSL) over TCP. For details, see the section called "Encrypting log messages with TLS".

When you send your log messages from a syslog-ng PE client through the network to a syslog-ng PE server, you can use different protocols and options. Every combination has its advantages and disadvantages. The most important thing is to use matching protocols and options, so the server handles the incoming log messages properly. For details, see the section called "Things to consider when forwarding messages between syslog-ng PE hosts".

Declaration:

```
network([options]);
```

By default, the *network()* driver binds to **0.0.0.0**, meaning that it listens on every available IPV4 interface on the TCP/601 port. To limit accepted connections to only one interface, use the *localip()* parameter. To listen on IPv6 addresses, use the **ip-protocol (6)** option.

Example 6.15. Using the network() driver

Using only the default settings: listen on every available IPV4 interface on the TCP/601 port.

```
source  s_network  {
       network();
 };
```

UDP source listening on `192.168.1.1` (the default port for UDP is 514):

```
source  s_network  {
       network(
             ip("192.168.1.1")
             transport("udp")
       );
 };
```

TCP source listening on the IPv6 localhost, port 2222:

```
source  s_network6  {
       network(
             ip("::1")
             transport("tcp")
             port(2222)
             ip-protocol(6)
       );
 };
```

A TCP source listening on a TLS-encrypted channel.

```
source  s_network  {
       network(
             transport("tcp")
             port(2222)
             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. For details, see the section called “Collecting messages using the IETF syslog protocol (syslog() driver)”.

```
source  s_tcp_syslog  {
       network(
             ip("127.0.0.1")
             flags(syslog-protocol)
       );
};
```

For details on the options of the *network()* source, see the section called "network() source options".

network() source options

The *network()* driver has the following options.

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. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

flags()

Type:	assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse, dont-store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The *assume-utf8* flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the *validate-utf8* flag.

dont-store-legacy-msghdr: By default, 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**). If you do not want to store the original header of the message, enable the *dont-store-legacy-msghdr* flag.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.

- *expect-hostname*: If the *expect-hostname* flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the *no-hostname* flag by default.

- *kernel*: The *kernel* flag makes the source default to the **LOG_KERN | LOG_NOTICE** priority if not specified otherwise.

- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the *rltp*, *syslog()*, *network()*, *unix-dgram()* drivers support multi-line messages.

- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the *flags(no-parse)* option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since *flags(no-parse)* disables message parsing, it interferes with other flags, for example, disables *flags(no-multi-line)*.

- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the *syslog* driver, which handles only messages that have a frame header.

- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section

called “IETF-syslog messages”). If the*BOM*[5]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

- *threaded*: The `threaded` flag enables multithreading for the source. For details on multithreading, see Chapter 18, *Multithreading and scaling in syslog-ng PE*.

  NOTE:

  The `syslog` source uses multiple threads only if the source uses the `tls` or `tcp` transport protocols.

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. By default, syslog-ng PE listens on every available interface. Note that this is not the address where messages are accepted from.

If you specify a multicast bind address and use the `udp` transport, syslog-ng PE automatically joins the necessary multicast group. TCP does not support multicasting.

ip-protocol()

Type:	number (IP version)
Default:	4

Description: Determines the internet protocol version of the given driver (`network()` or `syslog()`). The possible values are **4** and **6**, corresponding to IPv4 and IPv6. The default value is **4**.

ip-tos()

Type:	number (type of service)
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip-ttl()

Type:	number (hops)
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 forced to reload its configuration (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.

- If enabled (**keep-hostname(yes)**), syslog-ng PE assumes that the incoming log message was sent by the host specified in the *HOST* field of the message.
- If disabled (**keep-hostname(no)**), syslog-ng PE rewrites the *HOST* field of the message, either to the IP address (if the *use-dns()* parameter is set to **no**), or to the hostname (if the *use-dns()* parameter is set to **yes** and the IP address can be resolved to a hostname) of the host sending the message to syslog-ng PE. For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

NOTE:

If the log message does not contain a hostname in its *HOST* field, syslog-ng PE automatically adds a hostname to the message.

- For messages received from the network, this hostname is the address of the host that sent the message (this means the address of the last hop if the message was transferred via a relay).

- For messages received from the local host, syslog-ng PE adds the name of the host.

This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

When relaying messages, enable this option on the syslog-ng PE server and also on every relay, otherwise syslog-ng PE 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.

CAUTION:

To use the `S_` macros, the *`keep-timestamp()`* option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *`max-connections()`* option is set, the *`log-iw-size()`* will be divided by the number of connections, otherwise *`log-iw-size()`* is divided by 10 (the default value of the *`max-connections()`* option). The resulting number is the initial window size of each connection.

For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.16. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

max-connections()

Type:	number (simultaneous connections)
Default:	10

Description: Specifies the maximum number of simultaneous connections.

multi-line-garbage()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-garbage()* option when processing multi-line messages that contain unneeded parts between the messages. Specify a string or regular expression that matches the beginning of the unneeded message parts. If the *multi-line-garbage()* option is set, syslog-ng PE ignores the lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*. See also the *multi-line-prefix()* option.

When receiving multi-line messages from a source when the *multi-line-garbage()* option is set, but no matching line is received between two lines that match *multi-line-prefix()*, syslog-ng PE will continue to process the incoming lines as a single message until a line matching *multi-line-garbage()* is received.

CAUTION:

If the *multi-line-garbage()* option is set, syslog-ng PE discards lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

multi-line-prefix()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-prefix()* option to process multi-line messages, that is, log messages that contain newline characters (for example, Tomcat logs). Specify a string or regular expression that matches the beginning of the log messages. Use as simple regular expressions as possible, because complex regular expressions can severely reduce the rate of processing multi-line messages. If the *multi-line-prefix()* option is set, syslog-ng PE ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the *multi-line-garbage()* option.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

TIP:

- To make multi-line messages more readable when written to a file, use a template in the destination and instead of the *${MESSAGE}* macro, use the following: *$(indent-multi-line ${MESSAGE})*. This expression inserts a tab after every newline character (except when a tab is already present), indenting every line of the message after the first. For example:

```
destination d_file {
    file ("/var/log/messages"
        template("${ISODATE} ${HOST} $(indent-multi-line ${MESSAGE})\n")
);
};
```

- For details on using templates, see the section called “Templates and macros”.

- To actually convert the lines of multi-line messages to single line (by replacing the newline characters with whitespaces), use the *flags(no-multi-line)* option in the source.

Example 6.17. Processing Tomcat logs

The log messages of the Apache Tomcat server are a typical example for multi-line log messages. The messages start with the date and time of the query in the **`YYYY.MM.DD HH:MM:SS`** format, as you can see in the following example.

```
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
SEVERE: Catalina.start:
LifecycleException:  service.getName(): "Catalina";  Protocol handler start
failed: java.net.BindException: Address already in use<null>:8080
       at org.apache.catalina.connector.Connector.start(Connector.java:1138)
       at org.apache.catalina.core.StandardService.start
(StandardService.java:531)
       at org.apache.catalina.core.StandardServer.start
(StandardServer.java:710)
       at org.apache.catalina.startup.Catalina.start(Catalina.java:583)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:288)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.commons.daemon.support.DaemonLoader.start
(DaemonLoader.java:177)
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
INFO: Server startup in 1206 ms
2010.06.09. 12:45:08 org.apache.coyote.http11.Http11Protocol pause
INFO: Pausing Coyote HTTP/1.1 on http-8080
2010.06.09. 12:45:09 org.apache.catalina.core.StandardService stop
INFO: Stopping service Catalina
```

To process these messages, specify a regular expression matching the timestamp of the messages in the *`multi-line-prefix()`* option. Such an expression is the following:

```
source s_file{ file("/var/log/tomcat6/catalina.2010-06-09.log" follow-freq
(0) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]{2}\.") flags(no-parse));
 };
```

Note that the **`flags(no-parse)`** is needed to avoid syslog-ng PE trying to interpret the date in the message.

CAUTION:

If you receive messages using the UDP protocol, do not use multi-line processing. If every line of a multi-line message is received in the same UDP packet, everything is fine, but if a multi-line message is fragmented into multiple UDP packets, the order they are received (thus the way how they are processed) cannot be guaranteed, and causes problems.

pad-size()

Type:	number (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng PE 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. If *`pad-size()`* was given and the incoming message does not fit into *`pad-size()`*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough bytes
```

port() or localport()

Type:	number (port number)
Default:	In case of TCP transport: 601 In case of UDP transport: 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`.

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. For details, see the **socket(7)** manual page.

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. For details, see the **socket(7)** manual page.

so-rcvbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. For details, see the **socket(7)** manual page.

CAUTION:

When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *so-rcvbuf()* option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the *so-rcvbuf()* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *so-rcvbuf()* at least to `2 097 152` bytes.

so-sndbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket send buffer in bytes. For details, see the **socket(7)** manual page.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

transport()

Type:	rltp, udp, tcp, or tls
Default:	tcp

Description: Specifies the protocol used to receive messages from the source.

CAUTION:

When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *`so-rcvbuf()`*

option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the *`so-rcvbuf()`* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *`so-rcvbuf()`* at least to `2 097 152` bytes.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various options related to TLS encryption, for example, key/certificate files and trusted CA locations. TLS can be used only with tcp-based transport protocols. For details, see the section called "TLS options".

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`). The syslog-ng PE application 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.

NOTE:

This option has no effect if the *`keep-hostname()`* option is enabled (**`keep-hostname (yes)`**) and the message contains a hostname.

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.

NOTE:

This option has no effect if the *keep-hostname()* option is enabled (**keep-hostname (yes)**) and the message contains a hostname.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is **yes**, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[5] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

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 the section called "pipe() source options".

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`.

CAUTION:

- **It is not recommended to use *pipe()* on anything else than real pipes.**
- **By default, syslog-ng PE uses the `flags(no-hostname)` option for pipes, meaning that syslog-ng PE assumes that the log messages received from the pipe do not contain the hostname field. If your messages do contain the hostname field, use `flags(expect-hostname)`. For details, see the section called "flags()".**

Example 6.18. Using the pipe() driver

```
source s_pipe { pipe("/dev/pipe" pad-size(2048)); };
```

pipe() source options

The *pipe* driver has the following options:

flags()

Type: assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse,

dont-store-legacy-msghdr, syslog-protocol, validate-utf8

Default: empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The *assume-utf8* flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the *validate-utf8* flag.
- *dont-store-legacy-msghdr*: By default, 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**). If you do not want to store the original header of the message, enable the *dont-store-legacy-msghdr* flag.
- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.
- *expect-hostname*: If the *expect-hostname* flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the *no-hostname* flag by default.
- *kernel*: The *kernel* flag makes the source default to the **LOG_KERN | LOG_NOTICE** priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the *rltp*, *syslog()*, *network()*, *unix-dgram()* drivers support multi-line messages.
- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the *flags(no-parse)* option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE}

part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.

- *syslog-protocol*: The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the `syslog` driver, which handles only messages that have a frame header.

- *validate-utf8*: The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section called "IETF-syslog messages"). If the*BOM*[6]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 (seconds)
Default:	1

Description: Indicates that the source should be checked periodically. 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.

CAUTION:

To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *max-connections()* option is set, the *log-iw-size()* will be divided by the number of connections, otherwise *log-iw-size()* is divided by 10 (the default value of the *max-connections()* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.19. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

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 (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng PE 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. If *pad-size()* was given and the incoming message does not fit into *pad-size()*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough 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`.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example `time-zone ("Europe/Budapest")`), or as the timezone offset in +/-HH:MM format (for example `+01:00`). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is `yes`, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[6] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Receiving messages from external applications

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);
```

Example 6.20. Using the program() driver

```
source s_program { program("/etc/init.d/mydaemon"); };
```

NOTE:

The program is restarted automatically if it exits.

program() source options

The *program()* driver has the following options:

flags()

Type:	assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse, dont-store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The *assume-utf8* flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the *validate-utf8* flag.

dont-store-legacy-msghdr: By default, 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**). If you do not want to store the original header of the message, enable the *dont-store-legacy-msghdr* flag.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.

- *expect-hostname*: If the *expect-hostname* flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the *no-hostname* flag by default.

- *kernel*: The *kernel* flag makes the source default to the **LOG_KERN | LOG_NOTICE** priority if not specified otherwise.

- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the *rltp*, *syslog()*, *network()*, *unix-dgram()* drivers support multi-line messages.

- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the *flags(no-parse)* option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since *flags(no-parse)* disables message parsing, it interferes with other flags, for example, disables *flags(no-multi-line)*.

- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the *syslog* driver, which handles only messages that have a frame header.

- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section

called "IETF-syslog messages"). If the*BOM*[7]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

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.

CAUTION:

To use the `S_` macros, the *`keep-timestamp()`* option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *`max-connections()`* option is set, the *`log-iw-size()`* will be divided by the number of connections, otherwise *`log-iw-size()`* is divided by 10 (the default value of the *`max-connections()`* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *`flush-lines()`* option set in the destination of your clients.

Example 6.21. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

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 (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses

2048 bytes). The syslog-ng PE 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. If *pad-size()* was given and the incoming message does not fit into *pad-size()*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough 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`.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is **yes**, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[7] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Collecting messages from tables or relational database

The *sql()* driver collects messages from an SQL database. Currently the Microsoft SQL (MSSQL), MySQL, Oracle and PostgreSQL databases are supported.

The *sql()* driver has the following required parameters: *database()*, *host()*, *table()*, *type()*, *uid-column()*, *datetime-column()* or *date-column()* and *time-column()*.

Declaration:

```
sql(options);
```

Note that the *sql()* source driver has the following restrictions and limitations:

- The *sql()* source driver does not monitor rotated tables. Therefore, every source can follow only one table.
- Timestamps columns with timezone are not supported. The syslog-ng PE application will retrieve the timestamp from these columns, but without the timezone information.
- The *sql()* source driver ignores the *log-msg-size()* option, that is, messages read from the SQL database can be longer than the maximal message length set in the *log-msg-size()* option.
- There is an ID column that is the monotonically increasing unique ID of the monitored table. It is not possible to use more than one ID column as complex ID.

Example 6.22. Using a MySQL source

With the following configuration, syslog-ng PE reads the records from the *test_log* table ordered by the **id** column.

The *test_log* table has the following structure:

```
CREATE TABLE “test_log” (
    “id” BIGINT NOT NULL AUTO_INCREMENT,
    “timestamp” TIMESTAMP NOT NULL,
    “host” VARCHAR(64) NOT NULL,
    “program” VARCHAR(64) NOT NULL,
    “log” VARCHAR(1024) NOT NULL
    );
```

The matching syslog-ng PE configuration that reads this table is the following:

```
source s_mysql { sql(
                        type("mysql")
                        host("10.140.32.29")
                        username("user")
                        password("secret")
                        database("test_db")
                        table("test_log")
                        datetime-column("timestamp")
                        uid-column("id")
                        message-template("${.sql.id} ${.sql.log}")
                        program-template("${.sql.program}")
                        host-template("${.sql.host}")
                        fast-follow-mode(yes)

 );};
```

Supported SQL sources by platform

NOTE:

If a particular database is not supported on a platform you are using it, use the *sql()* source on a different syslog-ng PE host to retrieve the records remotely.

Table 6.3. Supported SQL sources by platform

Platform \ Database	MSSQL	MySQL	PgSQL	Oracle
AIX	✔	✔	✔	-
FreeBSD 8	✔	✔	✔	-
FreeBSD 9	✔	✔	✔	-
FreeBSD 10	✔	✔	✔	-
HP-UX 11v2_on IA64	✔	✔	-	-
Linux (linux_glibc236): CentOS 5	✔	✔	✔	✔
Linux (linux_glibc236): CentOS 6	✔	✔	✔	✔
Linux (linux_glibc236): CentOS 7	✔	✔	✔	✔
Linux (linux_glibc236): Debian 7	✔	✔	✔	✔
Linux (linux_glibc236): openSUSE 11	✔	✔	✔	✔
Linux (linux_glibc236): Oracle Linux 5	✔	✔	✔	✔
Linux (linux_glibc236): Oracle Linux 6	✔	✔	✔	✔
Linux (linux_glibc236): Oracle Linux 7	✔	✔	✔	✔
Linux (linux_glibc236): SLES 11	✔	✔	✔	✔
Linux (linux_glibc236): Red Hat EL 5	✔	✔	✔	✔

Platform \ Database	MSSQL	MySQL	PgSQL	Oracle
Linux (linux_glibc236): Red Hat EL 6	✔	✔	✔	✔
Linux (linux_glibc236): Red Hat EL 7	✔	✔	✔	✔
Linux (linux_glibc236): Ubuntu 12.04	✔	✔	✔	✔
Linux (linux_glibc236): Ubuntu 14.04	✔	✔	✔	✔
Solaris 10 on SPARC and SPARC64	✔	✔	✔	✔
Solaris 11 on SPARC and SPARC64	✔	✔	✔	✔
Solaris 10 on x86_64	✔	✔	-	✔
Solaris 11 on x86_64	✔	✔	-	✔

sql() source options

The *sql()* driver has the following options.

archive-query()

Type:	string
Default:	

Description: The SQL-like statement which is executed after syslog-ng PE has queried a batch of records (as set in the *log-fetch-limit()* option). This statement can be used for example to archive or delete the records processed by syslog-ng PE. Note that the user account that syslog-ng PE uses to access the database requires the appropriate privileges to execute the statement. If executing the statement fails, syslog-ng PE will log the error message returned by the database, and continue processing the other records.

For details on customizing queries, see the section called “Customizing SQL queries”.

CAUTION:

The syslog-ng PE application does not validate or limit the contents of customized queries. Consequently, queries performed with a user with write-access can potentially modify or even harm the database. Use customized queries with care, and only for your own responsibility.

Example 6.23. A sample archive query

The following statement deletes the records already retreived from the database if the table is read from the beginning.

```
archive-query("DELETE FROM test_logs WHERE id <= $last_read_uid")
```

columns()

Type:	string list
Default:	empty

Description: The list of the name of the columns that will be queried. The default value is empty, meaning that all of the columns will be queried.

Example 6.24. SQL source option columns

columns("id","date","message")

connect-query()

Type:	string
Default:	

Description: The SQL-like statement which is executed after syslog-ng PE has successfully connected to the database.

For details on customizing queries, see the section called “Customizing SQL queries”.

CAUTION:

The syslog-ng PE application does not validate or limit the contents of customized queries. Consequently, queries performed with a user with write-access can potentially modify or even harm the database. Use customized queries with care, and only for your own responsibility.

Example 6.25. A sample connect query

```
connect-query("SET COLLATION_CONNECTION='utf8_general_ci'")
```

database()

Type:	string
Default:	logs

Description: Name of the database that stores the logs. Macros cannot be used in database name. Also, when using an Oracle database, you cannot use the same `database()` settings in more than one destination.

date-column(col_name, [format])

Type:	date, string
Default:	

Description: The column containing the date of the logrecord. The format value has to be in strptime format. For details, see the strptime manual page (**man strptime**).

NOTE:

If the type of the column is string, this is a required parameter.

datetime-column(col_name, [format])

Type:	string
Default:	

The following column types are supported:

- *MySQL*: timestamp, datetime, int
- *PostgreSQL*: timestamp, int
- *Oracle*: timestamp, int
- *MSSQL*: datetime, int

Description: The column containing the timestamp. If the type is int, it is considered to contain a UNIX timestamp. The format value is required if the type is string, and has to be in strptime format. For details, see the strptime manual page (**man strptime**).

Example 6.26. SQL source option datetime-column(col_name, [format])

```
datetime("timestampcol", "%Y-%m-%d")
```

default-facility()

Type:	facility string
Default:	local0

Description: This parameter assigns a facility value to the messages received from the sql source.

default-priority()

Type:	priority string
Default:	info

Description: This parameter assigns an emergency level to the messages received from the sql source.

fast-follow-mode()

Type:	yes|no
Default:	yes

Description: If set to **yes**, syslog-ng PE reads the database table as fast as possible, until it reaches the last record. After this, it will execute only one query in *follow-freq()* time. If it is set to **no**, syslog-ng PE executes only one query in *follow-freq()* time.

fetch-query()

Type:	string
Default:	Default value is generated in running time of syslog-ng

Description: The SQL-like statement used to collect the records from the database.

NOTE:

If this parameter is defined, syslog-ng PE does not check or validate it whether it is correct. Ensure that the customized statements are correct.

For details on customizing queries, see the section called "Customizing SQL queries".

CAUTION:

The syslog-ng PE application does not validate or limit the contents of customized queries. Consequently, queries performed with a user with write-access can potentially modify or even harm the database. Use customized queries with care, and only for your own responsibility.

Example 6.27. A sample fetch query

```
fetch-query("SELECT * FROM $table WHERE id > $last_read_uid AND test_logs.log
LIKE '%ERROR%' ORDER BY $uid")
```

The default fetch queries are the following:

- *MSSQL*:

```
SELECT TOP $fetch_limit $columns FROM $table WHERE $uid > '$last_read_uid'
ORDER BY $uid
```

- *MySQL*:

```
SELECT $columns FROM $table WHERE $uid > '$last_read_uid' ORDER BY $uid LIMIT
0,$fetch_limit
```

- *Oracle*:

```
SELECT $columns FROM (SELECT $table.*, Row_Number() OVER (ORDER BY $uid)
FetchRow FROM $table WHERE $uid > '$last_read_uid' ) WHERE FetchRow BETWEEN 0
AND $fetch_limit
```

- *PostgreSQL*:

```
SELECT $columns FROM $table WHERE $uid > '$last_read_uid' ORDER BY $uid LIMIT
$fetch_limit
```

follow-freq()

Type:	number (seconds)
Default:	10

Indicates that the source should be checked periodically. This is useful for SQL sources which always indicate readability, even though no new records were appended. If this value is higher than zero, syslog-ng will not attempt to use *poll()* on the SQL source, but checks whether the SQL source changed every time the *follow-freq()* interval (in seconds) has elapsed. Floating-point numbers (for example **1.5**) can be used 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.

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**.

host-template()

Type:	string
Default:	Empty string

Description: The template for defining the HOST part of the message. If the *host-template()* option is not specified, the value of the *host()* option will be used in the HOST part of the message.

NOTE:

This option requires the option *keep-hostname()* to be enabled: **keep-hostname (yes)**.

log-fetch-limit()

Type:	
Default:	100

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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *max-connections()* option is set, the *log-iw-size()* will be divided by the number of connections, otherwise *log-iw-size()* is divided by 10 (the default value of the *max-connections()* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.28. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

max-uid-query()

Type:	string
Default:	"SELECT max($uid) FROM $table"

Description: Used for retrieving the ID of the last row (that is, the last row of the last fetch) from the database source.

NOTE:

If this parameter is defined, syslog-ng PE does not check or validate whether it is correct. Ensure that the customized statements are correct.

This option is mandatory if *read-old-records(no)* and *fetch-query()* is defined.

NOTE:

When *fetch-query()* is defined, the default value will be:

```
"SELECT MAX($uid) FROM ( <fetch-query()> )"
```

Following a restart or reload of syslog-ng PE, if no new data has arrived since the last fetch, the custom query defined in *fetch-query()* will fetch everything starting from the very beginning. To avoid this and retrieve only the ID of the last row, replace the above value with something similar:

```
"SELECT max($uid) FROM $table"
```

This option cannot be used if *read-old-records(yes)*.

message-template()

Type:	string
Default:	

Description: The alias of the *template()* parameter.

password()

Type:	string
Default:	n/a

Description: Password of the database user.

port()

Type:	number (port number)
Default:	1433 TCP for MSSQL, 3306 TCP for MySQL, 1521 for Oracle, and 5432 TCP for PostgreSQL

Description: The port number to connect to.

prefix()

Type:	string
Default:	".sql"

Description: This prefix will be added to the name of the macros created from the database columns.

Example 6.29. SQL source option prefix()

If a database column is called `column1`, and the prefix option is set as `prefix ("customprefix.")`, the macro for the column will be called `customprefix.column1`.

program-template()

Type:	string
Default:	Empty string

Description: The template for defining the PROGRAM part of the message. If not specified, the *PROGRAM* message part will be empty.

read-old-records()

Type:	yes|no
Default:	yes

Description:

- **If syslog-ng PE reads the database for the first time:**
  - If set to `yes`, syslog-ng PE starts reading the records from the beginning of the database.
  - If set to `no`, syslog-ng PE only reads the new records in the database.
- **If syslog-ng PE has read the database before:** If syslog-ng PE has read the database before, the `read-old-records()` option has created a persist entry during the first reading.
  - If set to `yes`, syslog-ng PE reads the `last_read_uid` from the persist file, and reads every record from the `sql` table where the `uid` is greater than the `last_read_uid`. Next, syslog-ng PE starts polling the database and updates the `last_read_uid` with the most recent `uid` from the database.
  - If set to `no`, syslog-ng PE first runs a `select max(uid)` SQL query for the table, then updates the `last_read_uid` with the value from the SQL query. After that, syslog-ng PE starts polling the database, starting from this updated `last_read_uid` value.

(missing or bad snippet)

table()

Type:	string
Default:	

Description: The name of the monitored table. Only a single literal name is accepted, macros cannot be used in the name of the table. Monitoring rotated tables is not supported.

table-init-query()

Type:	string
Default:	

Description: The SQL-like statement which is executed before fetching the first batch of records.

For details on customizing queries, see the section called “Customizing SQL queries”.

> **CAUTION:**
>
> **The syslog-ng PE application does not validate or limit the contents of customized queries. Consequently, queries performed with a user with write-access can potentially modify or even harm the database. Use customized queries with care, and only for your own responsibility.**

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.

template()

Type:	string
Default:	

Description: The template of the message (*`${MSG}`*) to be generated. If not specified, the following template will be used: **`"$(format-welf --key <prefix>*)"`** where <prefix> is the value of the *`prefix()`* option. This template converts the retrieved records into the WebTrends Enhanced Log file Format (WELF).

- For details on the WELF format, see https://www3.trustwave.com/support/kb/article.aspx?id=10899.
- For details on the *`format-welf()`* template function, see the section called "format-welf".

NOTE:

The *`format-welf`* function does not keep the order of columns between queries.

Example 6.30. SQL source option template()

Using the default template for a table that has two columns (**`id`** and **`message`**) The result of the first query is the following: **`'.sql.id=12 .sql.message="test message"'`**, whereas the result of the second query can be: **`'.sql.message="test message" .sql.id=12 '`**

time-column(col_name, [format])

Type:	time, string
Default:	

Description: The column containing the time of the logrecord. The format value has to be in strptime format.

NOTE:

If the type of the column is string, this is a required parameter.

time-reopen()

Accepted values:	number (seconds)
Default:	60

Description: The time to wait in seconds before a dead connection is reestablished.

time-zone()

Type:	timezone in +/-HH:MM format
Default:	

Description: The default timezone, if set. If this option is not set, the default timezone is the local timezone.

type()

Type:	mssql, mysql, oracle, or pgsql
Default:	

Description: Specifies the type of the database, that is, the DBI database driver to use. Use the *mssql* option to send logs to an MSSQL database. For details, see the examples of the databases on the following sections.

uid-column()

Type:	string
Default:	

Description: The monotonically increasing unique ID of the monitored table (for example auto_increment). This column must be a type where the greater (>) operation is interpreted.

NOTE:

The value of the first record of this column should not be 0: syslog-ng PE will skip this value.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is **yes**, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

username()

Type:	string
Default:	n/a

Description: Name of the database user.

Customizing SQL queries

Every query executed by the SQL source can be customized. These customized queries are similar to SQL statements, but in can also refer to syslog-ng PE-specific variables with the prefix **$**. For example, *$table* is the name of the table.

CAUTION:

The syslog-ng PE application does not validate or limit the contents of customized queries. Consequently, queries performed with a user with write-access can potentially modify or even harm the database. Use customized queries with care, and only for your own responsibility.

The available variables are the following:

NOTE:

The variables in the statement are not macro references.

- *$uid*: The name of the *uid-column()*.
- *$table*: The name of the *table()* option.
- *$last_read_uid*: The uid value of the last read record.
- *$columns*: Reach the *columns()* option of the SQL source (the default is *****). If columns is defined in the configuration (for example **columns("id" "message")**), then the value of this variable will be a comma separated list (for example **"id,message"**).
- *$fetch_limit*: Reach the value of the *log-fetch-limit()* option.

Example 6.31. SQL source fetch-query

The following queries records that are older than the last record:

```
SELECT * FROM $table WHERE $table.$uid > $last_read_uid ORDER BY $table.$uid
```

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 the section called "sun-streams() source options".

Declaration:

```
sun-streams(<name_of_the_streams_device> door(<filename_of_the_door>));
```

Example 6.32. Using the sun-streams() driver

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

sun-streams() source options

The *sun-streams()* driver has the following options.

door()

Type:	string
Default:	none

Description: Specifies the filename of a door to open, needed on Solaris above 2.5.1.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse, dont-store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *dont-store-legacy-msghdr*: By default, 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`**). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the **`LOG_KERN | LOG_NOTICE`** priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the `rltp`, `syslog()`, `network()`, `unix-dgram()` drivers support multi-line messages.
- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the `flags(no-parse)` option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.

- *syslog-protocol*: The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the `syslog` driver, which handles only messages that have a frame header.

  validate-utf8: The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section called "IETF-syslog messages"). If the*BOM*[8]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 (seconds)
Default:	1

Description: Indicates that the source should be checked periodically. 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.

CAUTION:

To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *max-connections()* option is set, the *log-iw-size()* will be divided by the number of connections, otherwise *log-iw-size()* is divided by 10 (the default value of the *max-connections()* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.33. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

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 (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng PE 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. If *pad-size()* was given and the incoming message does not fit into *pad-size()*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough 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`.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is **`yes`**, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[8] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Collecting messages using the IETF syslog protocol (syslog() driver)

The *syslog()* driver can receive messages from the network using the standard IETF-syslog protocol (as described in RFC5424-26). UDP, TCP, and TLS-encrypted TCP can all be used to transport the messages.

NOTE:

The *syslog()* driver can also receive BSD-syslog-formatted messages (described in RFC 3164, see the section called "BSD-syslog or legacy-syslog messages") if they are sent using the IETF-syslog protocol.

In syslog-ng PE versions 3.1 and earlier, the *syslog()* driver could handle only messages in the IETF-syslog (RFC 5424-26) format.

For the list of available optional parameters, see the section called "syslog() source options".

Declaration:

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

Example 6.34. 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. For details on the encryption settings, see the section called "TLS options".

```
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')
)
);};
```

CAUTION:

When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *so-rcvbuf()* option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the *so-rcvbuf()* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *so-rcvbuf()* at least to `2 097 152` bytes.

syslog() source options

The *syslog()* driver has the following options.

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. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

flags()

Type:	assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse, dont-store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *dont-store-legacy-msghdr*: By default, 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`**). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the **`LOG_KERN | LOG_NOTICE`** priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the `rltp`, `syslog()`, `network()`, `unix-dgram()` drivers support multi-line messages.
- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the `flags(no-parse)` option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.

syslog-protocol: The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the `syslog` driver, which handles only messages that have a frame header.

- *validate-utf8*: The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section called "IETF-syslog messages"). If the*BOM*[9]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

- *threaded*: The `threaded` flag enables multithreading for the source. For details on multithreading, see Chapter 18, *Multithreading and scaling in syslog-ng PE*.

  NOTE:

  The `syslog` source uses multiple threads only if the source uses the `tls` or `tcp` transport protocols.

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. By default, syslog-ng PE listens on every available interface. Note that this is not the address where messages are accepted from.

If you specify a multicast bind address and use the `udp` transport, syslog-ng PE automatically joins the necessary multicast group. TCP does not support multicasting.

ip-protocol()

Type:	number (IP version)
Default:	4

Description: Determines the internet protocol version of the given driver (`network()` or `syslog()`). The possible values are **4** and **6**, corresponding to IPv4 and IPv6. The default value is **4**.

ip-tos()

Type:	number (type of service)
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip-ttl()

Type:	number (hops)
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 forced to reload its configuration (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.

- If enabled (**keep-hostname(yes)**), syslog-ng PE assumes that the incoming log message was sent by the host specified in the *HOST* field of the message.
- If disabled (**keep-hostname(no)**), syslog-ng PE rewrites the *HOST* field of the message, either to the IP address (if the *use-dns()* parameter is set to **no**), or to the hostname (if the *use-dns()* parameter is set to **yes** and the IP address can be resolved to a hostname) of the host sending the message to syslog-ng PE. For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

NOTE:

If the log message does not contain a hostname in its *HOST* field, syslog-ng PE

automatically adds a hostname to the message.

- For messages received from the network, this hostname is the address of the host that sent the message (this means the address of the last hop if the message was transferred via a relay).
- For messages received from the local host, syslog-ng PE adds the name of the host.

This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

When relaying messages, enable this option on the syslog-ng PE server and also on every relay, otherwise syslog-ng PE 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.

CAUTION:

To use the `S_` macros, the *`keep-timestamp()`* option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *max-connections()* option is set, the *log-iw-size()* will be divided by the number of connections, otherwise *log-iw-size()* is divided by 10 (the default value of the *max-connections()* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.35. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

max-connections()

Type:	number (simultaneous connections)
Default:	10

Description: Specifies the maximum number of simultaneous connections.

multi-line-garbage()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-garbage()* option when processing multi-line messages that contain unneeded parts between the messages. Specify a string or regular expression that matches the beginning of the unneeded message parts. If the *multi-line-garbage()* option is set, syslog-ng PE ignores the lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*. See also the *multi-line-prefix()* option.

When receiving multi-line messages from a source when the *multi-line-garbage()* option is set, but no matching line is received between two lines that match *multi-line-prefix()*, syslog-ng PE will continue to process the incoming lines as a single message until a line matching *multi-line-garbage()* is received.

CAUTION:

If the *multi-line-garbage()* option is set, syslog-ng PE discards lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

multi-line-prefix()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-prefix()* option to process multi-line messages, that is, log messages that contain newline characters (for example, Tomcat logs). Specify a string or regular expression that matches the beginning of the log messages. Use as simple regular expressions as possible, because complex regular expressions can severely reduce the rate of processing multi-line messages. If the *multi-line-prefix()* option is set, syslog-ng PE ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the *multi-line-garbage()* option.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

TIP:

- To make multi-line messages more readable when written to a file, use a template in the destination and instead of the *${MESSAGE}* macro, use the following: *$(indent-multi-line ${MESSAGE})*. This expression inserts a tab after every newline character (except when a tab is already present), indenting every line of the message after the first. For example:

```
destination d_file {
    file ("/var/log/messages"
        template("${ISODATE} ${HOST} $(indent-multi-line ${MESSAGE})\n")
);
```

```
};
```

- For details on using templates, see the section called “Templates and macros”.

  To actually convert the lines of multi-line messages to single line (by replacing the newline characters with whitespaces), use the *flags(no-multi-line)* option in the source.

Example 6.36. Processing Tomcat logs

The log messages of the Apache Tomcat server are a typical example for multi-line log messages. The messages start with the date and time of the query in the **YYYY.MM.DD HH:MM:SS** format, as you can see in the following example.

```
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
SEVERE: Catalina.start:
LifecycleException:  service.getName(): "Catalina";  Protocol handler start
failed: java.net.BindException: Address already in use<null>:8080
       at org.apache.catalina.connector.Connector.start(Connector.java:1138)
       at org.apache.catalina.core.StandardService.start
(StandardService.java:531)
       at org.apache.catalina.core.StandardServer.start
(StandardServer.java:710)
       at org.apache.catalina.startup.Catalina.start(Catalina.java:583)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:288)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.commons.daemon.support.DaemonLoader.start
(DaemonLoader.java:177)
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
INFO: Server startup in 1206 ms
2010.06.09. 12:45:08 org.apache.coyote.http11.Http11Protocol pause
INFO: Pausing Coyote HTTP/1.1 on http-8080
2010.06.09. 12:45:09 org.apache.catalina.core.StandardService stop
INFO: Stopping service Catalina
```

To process these messages, specify a regular expression matching the timestamp of the messages in the *multi-line-prefix()* option. Such an expression is the following:

```
source s_file{ file("/var/log/tomcat6/catalina.2010-06-09.log" follow-freq
(0) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]{2}\.") flags(no-parse));
};
```

Note that the **flags(no-parse)** is needed to avoid syslog-ng PE trying to interpret the date in the message.

CAUTION:

If you receive messages using the UDP protocol, do not use multi-line processing. If every line of a multi-line message is received in the same UDP packet, everything is fine, but if a multi-line message is fragmented into multiple UDP packets, the order they are received (thus the way how they are processed) cannot be guaranteed, and causes problems.

pad-size()

Type:	number (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng PE 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. If *pad-size()* was given and the incoming message does not fit into *pad-size()*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough bytes
```

port() or localport()

Type:	number (port number)
Default:	In case of TCP transport: 601 In case of UDP transport: 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`.

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. For details, see the **socket(7)** manual page.

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. For details, see the **socket(7)** manual page.

so-rcvbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. For details, see the **socket(7)** manual page.

CAUTION:

When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *`so-rcvbuf()`* option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

> **As a general rule, increase the *so-rcvbuf()* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *so-rcvbuf()* at least to `2 097 152` bytes.**

so-sndbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket send buffer in bytes. For details, see the **socket(7)** manual page.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

transport()

Type:	rltp, udp, tcp, or tls
Default:	tcp

Description: Specifies the protocol used to receive messages from the source.

CAUTION:

When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *so-rcvbuf()* option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the *so-rcvbuf()* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *so-rcvbuf()* at least to `2 097 152` bytes.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various options related to TLS encryption, for example, key/certificate files and trusted CA locations. TLS can be used only with tcp-based transport protocols. For details, see the section called "TLS options".

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`). The syslog-ng PE application 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.

NOTE:

This option has no effect if the *keep-hostname()* option is enabled (**keep-hostname (yes)**) and the message contains a hostname.

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.

NOTE:

This option has no effect if the *keep-hostname()* option is enabled (**keep-hostname (yes)**) and the message contains a hostname.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is **yes**, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[9] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Collecting the system-specific log messages of a platform

Starting with version 4 F1, syslog-ng PE can automatically collect the system-specific log messages of the host on a number of platforms using the *system()* driver. If the *system()* driver is included in the syslog-ng PE configuration file, syslog-ng PE automatically adds the following sources to the syslog-ng PE configuration.

NOTE:

syslog-ng PE versions 4.1-5.0 used an external script to generate the *system()* source, but this was problematic in certain situations, for example, when the host used a strict AppArmor profile. Therefore, the *system()* source is now generated internally in syslog-ng PE.

The *system()* driver is also used in the default configuration file of syslog-ng PE. For details on the default configuration file, see Example 4.1, "The default configuration file of syslog-ng PE".

CAUTION:

If syslog-ng PE does not recognize the platform it is installed on, it does not add any sources.

Table 6.4. Sources automatically added by syslog-ng Premium Edition

Platform	Message source
AIX	`unix-dgram("/dev/log");`
	`unix-dgram("/var/run/log");`
FreeBSD	`unix-dgram("/var/run/logpriv" perm(0600));`
	`file("/dev/klog" follow-freq(0) program-override("kernel") flags(no-parse));`
HP-UX	`pipe("/dev/log" pad-size(2048));`
Linux	`unix-dgram("/dev/log");`
	`file("/proc/kmsg" program-override("kernel") flags(kernel));`
Solaris 8	`sun-streams("/dev/log");`
Solaris 9	`sun-streams("/dev/log" door("/etc/.syslog_door"));`

Platform	Message source
Solaris 10	`sun-streams("/dev/log" door("/var/run/syslog_door"));`

Collecting messages from the systemd-journal system log storage

The *systemd-journal()* source is used on various Linux distributions, such as RHEL (from RHEL7) and CentOS. The *systemd-journal()* source driver can read the structured name-value format of the journald system service, making it easier to reach the custom fields in the message.

The *systemd-journal()* source driver is designed to read only local messages through the systemd-journal API. It is not possible to set the location of the journal files, or the directories.

NOTE:

The *log-msg-size()* option is not applicable for this source. Use the *max-field-size()* option instead.

NOTE:

This source will not handle the following cases:

- corrupted journal file
- incorrect journal configuration
- any other journald-related bugs

NOTE:

If you are using RHEL-7, the default source in the configuration is *systemd-journal()* instead of **unix-dgram("/dev/log")** and **file("/proc/kmsg")**. If you are using **unix-dgram("/dev/log")** or **unix-stream("/dev/log")** in your configuration as a source, syslog-ng PE will revert to using *systemd-journal()* instead.

CAUTION:

Only one *systemd-journal()* source can be configured in the configuration file. If there are more than one *systemd-journal()* sources configured, syslog-ng PE will not start.

Declaration:

```
systemd-journal(options);
```

Example 6.37. Sending all fields through syslog protocol using the systemd-journal() driver

To send all fields through the syslog protocol, enter the prefix in the following

format: "**.SDATA.<name>**".

```
@version: 6.0

source s_journald {
       systemd-journal(prefix(".SDATA.journald."));
};

destination d_network {
       syslog("server.host");
};

log {
       source(s_journald);
       destination(d_network);
};
```

Example 6.38. Filtering for a specific field using the systemd-journal() driver

```
@version: 6.0

source s_journald {
       systemd-journal(prefix(".SDATA.journald."));
};

filter f_uid {"${.SDATA.journald._UID}" eq "1000"};

destination d_network {
       syslog("server.host");
};

log {
       source(s_journald);
       filter(f_uid);
       destination(d_network);
};
```

Example 6.39. Sending all fields in value-pairs using the systemd-journal() driver

```
@version: 6.0

source s_local {
       systemd-journal(prefix("journald."));
};

destination d_network {
       network("server.host" template("$(format_json --scope rfc5424 --key
journald.*)\n"));
};

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

The journal contains credential information about the process that sent the log message. The syslog-ng PE application makes this information available in the following macros:

Journald field	syslog-ng predefined macro
MESSAGE	$MESSAGE
_HOSTNAME	$HOST
_PID	$PID
_COMM if does not exist SYSLOG_IDENTIFIER	$PROGRAM
SYSLOG_FACILITY	$FACILITY_NUM
PRIORITY	$LEVEL_NUM

systemd-journal() source options

The *systemd-journal()* driver has the following options:

default-facility()

Type:	facility string
Default:	local0

Description: The default facility value if the SYSLOG_FACILITY entry does not exist.

default-level()

Type:	string
Default:	notice

Description: The default level value if the PRIORITY entry does not exist.

host-override()

Type:	string
Default:	

Description: Replaces the ${HOST} part of the message with the parameter string.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (**`keep-hostname(yes)`**), syslog-ng PE will retain the hostname information read from the systemd journal messages.
- If disabled (**`keep-hostname(no)`**), syslog-ng PE will use the hostname that has been set up for the operating system instance that syslog-ng is running on. To query or set this value, use the **hostnamectl** command.

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 (messages)
Default:	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.

max-field-size()

Type:	number (characters)
Default:	65536

Description: The maximum length of a field's value.

prefix()

Type:	string
Default:	

Description: If this option is set, every non-built-in mapped names get a prefix (for example: `".SDATA.journald."`).

time-zone()

Type:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

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.

> NOTE:
>
> This option has no effect if the *`keep-hostname()`* option is enabled (**`keep-hostname (yes)`**) and the message contains a hostname.

Collecting messages from remote hosts using the BSD syslog protocol

NOTE:

The *tcp()*, *tcp6()*, *udp()*, and *udp6()* drivers are obsolete. Use the *network()* source and the *network()* destination instead. For details, see the section called "Collecting messages using the RFC3164 protocol (network() driver)" and the section called "Sending messages to a remote log server using the RFC3164 protocol (network() driver)", respectively.

The *tcp()*, *tcp6()*, *udp()*, *udp6()* drivers can receive syslog messages conforming to RFC3164 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.

To convert your existing *tcp()*, *tcp6()*, *udp()*, *udp6()* source drivers to use the *network()* driver, see Procedure 6.1, "Change an old source driver to the network() driver".

tcp(), tcp6(), udp() and udp6() source options — OBSOLETE

NOTE:

The *tcp()*, *tcp6()*, *udp()*, and *udp6()* drivers are obsolete. Use the *network()* source and the *network()* destination instead. For details, see the section called "Collecting messages using the RFC3164 protocol (network() driver)" and the section called "Sending messages to a remote log server using the RFC3164 protocol (network() driver)", respectively.

To convert your existing *tcp()*, *tcp6()*, *udp()*, *udp6()* source drivers to use the *network()* driver, see Procedure 6.1, "Change an old source driver to the network() driver".

Procedure 6.1. Change an old source driver to the network() driver

To replace your existing *tcp()*, *tcp6()*, *udp()*, *udp6()* sources with a *network()* source, complete the following steps.

1. Replace the driver with **network**. For example, replace **udp(** with **network(**
2. Set the transport protocol.
   - If you used TLS-encryption, add the **transport("tls")** option, then continue with the next step.
   - If you used the **tcp** or **tcp6** driver, add the **transport("tcp")** option.
   - If you used the **udp** or **udp** driver, add the **transport("udp")** option.
3. If you use IPv6 (that is, the **udp6** or **tcp6** driver), add the **ip-protocol("6")** option.

4. If you did not specify the port used in the old driver, check the section called "network() source options" and verify that your clients send the messages to the default port of the transport protocol you use. Otherwise, set the appropriate port number in your source using the *port()* option.
5. All other options are identical. Test your configuration with the **syslog-ng --syntax-only** command.

   The following configuration shows a simple **tcp** source.

```
source  s_old_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')
              )
       );
};
```

   When replaced with the **network()** driver, it looks like this.

```
source  s_new_network_tcp  {
       network(
              transport("tls")
              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')
              )
       );
};
```

Collecting systemd messages using a socket

On platforms running systemd, the *systemd-syslog()* driver reads the log messages of systemd using the `/run/systemd/journal/syslog` socket. Note the following points about this driver:

- If possible, use the more reliable systemd-journal() driver instead.
- The socket activation of systemd is buggy, causing some log messages to get lost during system startup.
- If syslog-ng PE is running in a jail or a Linux Container (LXC), it will not read from the `/dev/kmsg` or `/proc/kmsg` files.

Declaration:

```
systemd-syslog();
```

Example 6.40. Using the systemd-syslog() driver

```
@version: 6.0

source s_systemdd {
       systemd-syslog();
};

destination d_network {
       syslog("server.host");
};

log {
       source(s_systemdd);
       destination(d_network);
};
```

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 the section called "unix-stream() and unix-dgram() source options"

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.

Example 6.41. 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"); };
```

unix-stream() and unix-dgram() source options

These two drivers behave similarly: they open an *AF_UNIX* socket and start listening on it for messages. The following options can be specified for these drivers:

create-dirs()

Type:	yes or no
Default:	no

Description: Enable creating non-existing directories when creating the socket files.

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. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

flags()

Type:	assume-utf8, empty-lines, expect-hostname, kernel, no-multi-line, no-parse, dont-store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The *`assume-utf8`* flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the *`validate-utf8`* flag.
- *dont-store-legacy-msghdr*: By default, 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`**). If you do not want to store the original header of the message, enable the *`dont-store-legacy-msghdr`* flag.
- *empty-lines*: Use the *`empty-lines`* flag to keep the empty lines of the messages. By default, syslog-ng PE removes empty lines automatically.
- *expect-hostname*: If the *`expect-hostname`* flag is enabled, syslog-ng PE will assume that the log message contains a hostname and parse the message accordingly. This

- is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the **`LOG_KERN | LOG_NOTICE`** priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way syslog-ng PE assumes that the first part of the message header is ${PROGRAM} instead of ${HOST}. For example:

  ```
  source s_dell { network(port(2000) flags(no-hostname)); };
  ```
- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line. Note that this happens only if the underlying transport method actually supports multi-line messages. Currently the `rltp`, `syslog()`, `network()`, `unix-dgram()` drivers support multi-line messages.
- *no-parse*: By default, syslog-ng PE parses incoming messages as syslog messages. The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. The syslog-ng PE application will generate a new syslog header (timestamp, host, and so on) automatically and put the entire incoming message into the MSG part of the syslog message. This flag is useful for parsing messages not complying to the syslog format.

  If you are using the `flags(no-parse)` option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *syslog-protocol*: The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard (RFC5424), but without the frame header. Note that this flag is not needed for the `syslog` driver, which handles only messages that have a frame header.
- *validate-utf8*: The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see the section called “IETF-syslog messages”). If the*BOM*[10]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

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.

CAUTION:

To use the s_ macros, the *keep-timestamp()* option must be enabled (this is the default behavior of syslog-ng PE).

log-fetch-limit()

Type:	number (messages)
Default:	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 (messages)
Default:	1000

Description: The size of the initial window, this value is used during flow control. If the *max-connections()* option is set, the *log-iw-size()* will be divided by the number of connections, otherwise *log-iw-size()* is divided by 10 (the default value of the *max-connections()* option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng PE clients, make sure that the window size is larger than the *flush-lines()* option set in the destination of your clients.

Example 6.42. Initial window size of a connection

If **log-iw-size(1000)** and **max-connections(10)**, then each connection will have an initial window size of 100.

log-msg-size()

Type:	number (bytes)
Default:	Use the global *log-msg-size()* option, which defaults to **65535**.

Description: Specifies the maximum length of incoming log messages. Uses the value of the global option if not specified. For details on how encoding affects the size of the message, see the section called "Message size and encoding".

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 (simultaneous connections)
Default:	256

Description: Limits the number of simultaneously open connections. Cannot be used with *unix-dgram()*.

multi-line-garbage()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-garbage()* option when processing multi-line messages that contain unneeded parts between the messages. Specify a string or regular expression that matches the beginning of the unneeded message parts. If the *multi-line-garbage()* option is set, syslog-ng PE ignores the lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*. See also the *multi-line-prefix()* option.

When receiving multi-line messages from a source when the *multi-line-garbage()* option is set, but no matching line is received between two lines that match *multi-line-prefix()*, syslog-ng PE will continue to process the incoming lines as a single message until a line matching *multi-line-garbage()* is received.

CAUTION:

If the *multi-line-garbage()* option is set, syslog-ng PE discards lines between the line matching the *multi-line-garbage()* and the next line matching *multi-line-prefix()*.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

This option is not available for the *unix-dgram* driver.

multi-line-prefix()

Type:	regular expression
Default:	empty string

Description: Use the *multi-line-prefix()* option to process multi-line messages, that is, log messages that contain newline characters (for example, Tomcat logs). Specify a string or regular expression that matches the beginning of the log messages. Use as simple regular expressions as possible, because complex regular expressions can severely reduce the rate of processing multi-line messages. If the *multi-line-prefix()* option is set, syslog-ng PE ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the *multi-line-garbage()* option.

NOTE:

Starting with syslog-ng PE version 3.2.1, a message is considered complete if no new lines arrive to the message for 10 seconds, even if no line matching the *multi-line-garbage()* option is received.

TIP:

- To make multi-line messages more readable when written to a file, use a template in the destination and instead of the *${MESSAGE}* macro, use the following: *$(indent-multi-line ${MESSAGE})*. This expression inserts a tab after every newline character (except when a tab is already present), indenting every line of the message after the first. For example:

  ```
  destination d_file {
      file ("/var/log/messages"
          template("${ISODATE} ${HOST} $(indent-multi-line ${MESSAGE})\n")
  );
  };
  ```

- For details on using templates, see the section called “Templates and macros”.

- To actually convert the lines of multi-line messages to single line (by replacing the newline characters with whitespaces), use the *flags(no-multi-line)* option in the source.

Example 6.43. Processing Tomcat logs

The log messages of the Apache Tomcat server are a typical example for multi-line log messages. The messages start with the date and time of the query in the **YYYY.MM.DD HH:MM:SS** format, as you can see in the following example.

```
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
SEVERE: Catalina.start:
LifecycleException:  service.getName(): "Catalina";  Protocol handler start
failed: java.net.BindException: Address already in use<null>:8080
       at org.apache.catalina.connector.Connector.start(Connector.java:1138)
       at org.apache.catalina.core.StandardService.start
(StandardService.java:531)
       at org.apache.catalina.core.StandardServer.start
(StandardServer.java:710)
       at org.apache.catalina.startup.Catalina.start(Catalina.java:583)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:288)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
       at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
       at sun.reflect.DelegatingMethodAccessorImpl.invoke
```

```
(DelegatingMethodAccessorImpl.java:25)
       at java.lang.reflect.Method.invoke(Method.java:597)
       at org.apache.commons.daemon.support.DaemonLoader.start
(DaemonLoader.java:177)
2010.06.09. 12:07:39 org.apache.catalina.startup.Catalina start
INFO: Server startup in 1206 ms
2010.06.09. 12:45:08 org.apache.coyote.http11.Http11Protocol pause
INFO: Pausing Coyote HTTP/1.1 on http-8080
2010.06.09. 12:45:09 org.apache.catalina.core.StandardService stop
INFO: Stopping service Catalina
```

To process these messages, specify a regular expression matching the timestamp of the messages in the *multi-line-prefix()* option. Such an expression is the following:

```
source s_file{ file("/var/log/tomcat6/catalina.2010-06-09.log" follow-freq
(0) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]{2}\.") flags(no-parse));
 };
```

Note that the **flags(no-parse)** is needed to avoid syslog-ng PE trying to interpret the date in the message.

This option is not available for the *unix-dgram* driver.

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 (bytes)
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng PE 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. If *pad-size()* was given and the incoming message does not fit into *pad-size()*, syslog-ng will not read anymore from this pipe and displays the following error message:

```
Padding was set, and couldn't read enough bytes
```

perm()

Type:	number (octal notation)
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.

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. For details, see the **socket(7)** manual page.

so-rcvbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. For details, see the **socket(7)** manual page.

CAUTION:

When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *so-rcvbuf()* option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the *so-rcvbuf()* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *so-rcvbuf()* at least to `2 097 152` bytes.

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:	name of the timezone, or the timezone offset
Default:	

Description: The default timezone for messages read from the source. Applies only if no timezone is specified within the message itself.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

use-syslogng-pid()

Type:	yes or no
Default:	no

Description: If the value of this option is `yes`, then the PID value of the message will be overridden with the PID of the running syslog-ng process.

---

[10] The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

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 7.1. 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 { network("10.1.2.3" port(1999)); };
```

If name resolution is configured, you can use the hostname of the target server as well.

```
destination d_tcp { network("target_host" port(1999)); };
```

CAUTION:

- **Do not define the same drivers with the same parameters more than once, because it will cause problems. For example, do not open the same file in multiple destinations.**
- **Do not use the same destination in different log paths, because it can cause problems with most destination types. Instead, use filters and log paths to avoid such situations.**
- **Sources and destinations are initialized only when they are used in a log statement. For example, syslog-ng PE starts listening on a port or starts polling a file only if the source is used in a log statement. For details on creating log statements, see Chapter 8, *Routing messages: log paths, reliability, and filters*.**
- **Hazard of data loss! If your log files are on an NFS-mounted network file system, see the section called "NFS file system for log files".**

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

Table 7.1. Destination drivers available in syslog-ng

Name	Description
elasticsearch and elasticsearch2	Sends messages to an Elasticsearch server. The *elasticsearch2* driver supports Elasticsearch version 2.0-2.4.6.
file()	Writes messages to the specified file.
hdfs	Sends messages into a file on a Hadoop Distributed File System (HDFS) node.
kafka	Publishes log messages to the Apache Kafka message bus, where subscribers can access them.
logstore()	Writes messages to the specified binary logstore file.
mongodb()	Sends messages to a MongoDB database.
network()	Sends messages to a remote host using the BSD-syslog protocol over IPv4 and IPv6. Supports the TCP, UDP,RLTP™, and TLS network protocols.
pipe()	Writes messages to the specified named pipe.
program()	Forks and launches the specified program, and sends messages to its standard input.
smtp()	Sends messages send mail to trusted recipients, through a controlled channel using the SMTP protocol.
snmp()	Sends messages to the specified remote host using the SNMP v2c or v3 protocol.
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*.
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.
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.

Sending messages directly to Elasticsearch version 2.0-2.4.6

Starting with version 5.6 of syslog-ng PE can directly send log messages to Elasticsearch, allowing you to search and analyze your data in real time, and visualize it with Kibana.

NOTE:

In order to use this destination, syslog-ng Premium Edition must run in server mode. Typically, only the central syslog-ng Premium Edition server uses this destination. For details on the server mode, see the section called "Server mode".

Note the following limitations when using the syslog-ng PE*elasticsearch2* destination:

- This destination is only supported on the Linux platforms that use the `linux glibc2.11` installer, including: Red Hat ES 7, Ubuntu 14.04 (Trusty Tahr).
- Since syslog-ng PE uses the official Java Elasticsearch libraries, the *elasticsearch2* destination has significant memory usage.
- The log messages of the underlying client libraries are available in the *internal()* source of syslog-ng PE.

Declaration:

```
@module mod-java
@include "scl.conf"

elasticsearch2(
      index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
      type("test")
      cluster("syslog-ng")
);
```

Example 7.5. Sending log data to Elasticsearch version 2.0-2.4.6

The following example defines an *elasticsearch2* destination that sends messages in transport mode to an Elasticsearch server running on the localhost, using only the required parameters.

```
@module mod-java
@include "scl.conf"

destination d_elastic {
   elasticsearch2(
      index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
      type("test")
   );
};
```

The following example sends 10000 messages in a batch, in node mode, and includes a custom unique ID for each message.

```
@module mod-java
@include "scl.conf"

options {
   threaded(yes);
   use_uniqid(yes);
};

source s_syslog {
   syslog();
};

destination d_elastic {
   elasticsearch2(
      index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
      type("test")
      cluster("syslog-ng")
      client_mode("node")
      custom_id("${UNIQID}")
      flush-limit("10000")
   );
};

log {
   source(s_syslog);
   destination(d_elastic);
   flags(flow-control);
};
```

- To install the software required for the *elasticsearch2* destination, see Procedure 7.2, “Prerequisites”.

For details on how the *elasticsearch2* destination works, see the section called "How syslog-ng PE interacts with Elasticsearch".

- For the list of options, see the section called "Elasticsearch destination options".

Procedure 7.2. Prerequisites

To send messages from syslog-ng PE to Elasticsearch, complete the following steps.

Steps:

1. Download and install the Java Runtime Environment (JRE), 2.x (or newer). The syslog-ng PE*elasticsearch2* destination is tested and supported when using the Oracle implementation of Java. Other implementations are untested and unsupported, they may or may not work as expected.
2. Download the Elasticsearch libraries (version 2.0-2.4.6) from https://www.elastic.co/downloads/elasticsearch.One Identity tests the destination using Elasticsearch version 2.4..
3. Extract the Elasticsearch libraries into a temporary directory, then collect the various `.jar` files into a single directory (for example, `/opt/elasticsearch/lib/`) where syslog-ng PE can access them. You must specify this directory in the syslog-ng PE configuration file. The files are located in the `lib` directory and its subdirectories of the Elasticsearch release package.

How syslog-ng PE interacts with Elasticsearch

The syslog-ng PE application sends the log messages to the official Elasticsearch client library, which forwards the data to the Elasticsearch nodes. The way how syslog-ng PE interacts with Elasticsearch is described in the following steps.

- After syslog-ng PE is started and the first message arrives to the *elasticsearch2* destination, the *elasticsearch2* destination tries to connect to the Elasticsearch server or cluster. If the connection fails, syslog-ng PE will repeatedly attempt to connect again after the period set in *time-reopen()* expires.
- If the connection is established, syslog-ng PE sends JSON-formatted messages to Elasticsearch.
  - If *flush_limit* is set to 1: syslog-ng PE sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, syslog-ng PE repeats sending the message, as set in the *retries()* parameter. If sending the message fails for *retries()* times, syslog-ng PE drops the message.

    This method ensures reliable message transfer, but is slow.

If *flush_limit* is higher than 1: syslog-ng PE sends messages in a batch, and receives the response asynchronously. In case of a problem, syslog-ng PE cannot resend the messages.

This method is relatively fast (depending on the size of *flush_limit*), but the transfer is not reliable. In transport mode, several messages can be lost before syslog-ng PE recognizes the error. Node mode is more reliable in this sense, because the message loss rate is significantly lower.

If *concurrent-requests* is higher than 1, syslog-ng PE can send multiple batches simultaneously, increasing performance (and also the number of messages that can be lost in case of an error). For details, see the section called "concurrent_requests()".

Client modes

The syslog-ng PE application can interact with Elasticsearch in transport mode or node mode.

- **Transport mode.** The syslog-ng PE application uses the transport client API of Elasticsearch, and uses the *server()*, *port()*, and *cluster()* options from the syslog-ng PE configuration file.
- **Node mode.** The syslog-ng PE application acts as an Elasticsearch node (client no-data), using the node client API of Elasticsearch. Further options for the node can be describe in an Elasticsearch configuration file specified in the *resource()* option.

  NOTE:

  In Node mode, it is required to define the home of the elasticsearch installation with the *path.home* paramter in the `.yml` file. For example: **`path.home: /usr/share/elasticsearch`**.
- **Shield mode.** This mode is available only from syslog-ng PE version 5.6. In this mode, syslog-ng PE uses the transport client API of Elasticsearch, and uses the *server()*, *port()*, and *cluster()* options from the syslog-ng PE configuration file, but with Shield (X-Pack security) support. For more details about Shield, see: https://www.elastic.co/products/x-pack/security.

Elasticsearch destination options

The *elasticsearch2* destination can directly send log messages to Elasticsearch, allowing you to search and analyze your data in real time, and visualize it with Kibana. The *elasticsearch2* destination has the following options.

Required options:

The following options are required: *index()*, *type()*. In node mode, either the *cluster()* or the *resource()* option is required as well. Note that to use *elasticsearch2*, you must add the following lines to the beginning of your syslog-ng PE configuration:

```
@module mod-java
@include "scl.conf"
```

client_lib_dir()

Type:	string
Default:	N/A

Description: Include the path to the directory where you copied the required libraries (see Procedure 7.2, "Prerequisites"), for example, **client_lib_dir (/user/share/elasticsearch-2.2.0/lib)**.

client_mode()

Type:	transport | node | shield
Default:	node

Description: Specifies the client mode used to connect to the Elasticsearch server, for example, **client_mode("node")**.

- **Transport mode.** The syslog-ng PE application uses the transport client API of Elasticsearch, and uses the *server()*, *port()*, and *cluster()* options from the syslog-ng PE configuration file.
- **Node mode.** The syslog-ng PE application acts as an Elasticsearch node (client no-data), using the node client API of Elasticsearch. Further options for the node can be describe in an Elasticsearch configuration file specified in the *resource()* option.

  > NOTE:
  >
  > In Node mode, it is required to define the home of the elasticsearch installation with the *path.home* paramter in the *.yml* file. For example: **path.home: /usr/share/elasticsearch**.

- **Shield mode.** This mode is available only from syslog-ng PE version 5.6. In this mode, syslog-ng PE uses the transport client API of Elasticsearch, and uses the *server()*, *port()*, and *cluster()* options from the syslog-ng PE configuration file, but with Shield (X-Pack security) support. For more details about Shield, see: https://www.elastic.co/products/x-pack/security.
- To use this mode, add the Shield .jar file (shield-x.x.x.jar) to the same directory where your Elasticsearch .jar files are located. You can download the Shield

distribution and extract the .jar file manually, or you can get it from the Elasticsearch Maven repository.

It inherits the Transport mode options, but the Shield-related options must be configured in the .yml file (see the resource() option of syslog-ng PE). For more details about the possible options, see: https://www.elastic.co/guide/en/shield/current/reference.html#ref-ssl-tls-settings.

Example 7.6. Example for the .yml file

```
shield.user: es_admin:********
shield.transport.ssl: true
shield.ssl.keystore.path: /usr/share/elasticsearch/node.jks
shield.ssl.keystore.password: mypassword
```

cluster()

Type:	string
Default:	N/A

Description: Specifies the name or the Elasticsearch cluster, for example, **`cluster("my-elasticsearch-cluster")`**. Optionally, you can specify the name of the cluster in the Elasticsearch resource file. For details, see the section called “resource()”.

concurrent_requests()

Type:	number
Default:	0

Description: The number of concurrent (simultaneous) requests that syslog-ng PE sends to the Elasticsearch server. Set this option to 1 or higher to increase performance. When using the *`concurrent_requests()`* option, make sure that the *`flush-limit()`* option is higher than one, otherwise it will not have any noticeable effect. For details, see the section called “flush_limit()”.

CAUTION:

Hazard of data loss! Using the *`concurrent-requests()`* option increases the number of messages lost in case the Elasticsearch server becomes unaccessible.

custom_id()

Type:	template or template function
Default:	N/A

Description: Use this option to specify a custom ID for the records inserted into Elasticsearch. If this option is not set, the Elasticsearch server automatically generates and ID for the message. For example: **`custom_id(${UNIQID})`** (Note that to use the **`${UNIQID}`** macro, the *`use-uniqid()`* global option must be enabled. For details, see the section called "use-uniqid()".)

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()

Type:	yes|no
Default:	no

Description: If set to **`yes`**, syslog-ng PE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to **`no`**, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of *`reliable()`* option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored. This option has priority over *`--qdisk-dir=`*.

CAUTION:

When creating a new *`dir()`* option for a disk buffer, or modifying an existing one, make

sure you delete the persist file, or at least remove the relevant persist-entry.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file or the relevant entry is not deleted after modifying the *dir()* option, then following a restart, syslog-ng PE will look for or create disk-buffer files in their old location. To ensure that syslog-ng PE uses the new *dir()* setting, the persist file must not contain any information about the destinations which the disk-buffer file in question belongs to.

disk-buf-size()

Type:	number (bytes)
Default:	
	Description: This is a required option. The maximum size of the disk-buffer in bytes. The minimum value is 1048576 bytes. If you set a smaller value, the minimum value will be used automatically. It replaces the old *log-disk-fifo-size()* option.

mem-buf-length()

Type:	number (messages)
Default:	10000
	Description: Use this option if the option *reliable()* is set to **no**. This option contains the number of messages stored in overflow queue. It replaces the old *log-fifo-size()* option. It inherits the value of the global *log-fifo-size()* option if provided. If it is not provided, the default value is **10000** messages. Note that this option will be ignored if the option *reliable()* is set to **yes**.

mem-buf-size()

Type:	number (bytes)
Default:	163840000
	Description: Use this option if the option *reliable()* is set to **yes**. This option contains the size of the messages in bytes that is used in the memory part of the disk buffer. It

	replaces the old *log-fifo-size()* option. It does not inherit the value of the global *log-fifo-size()* option, even if it is provided. Note that this option will be ignored if the option *reliable()* is set to **no**.

quot-size()	
Type:	number (messages)
Default:	64
	Description: The number of messages stored in the output buffer of the destination.

Options *reliable()* and *disk-buf-size()* are required options.

Example 7.7. Examples for using disk-buffer()

In the following case reliable disk-buffer() is used.

```
destination  d_demo  {
       network(
                     "127.0.0.1"
                     port(3333)
                     disk-buffer(
                            mem-buf-size(10000)
                            disk-buf-size(2000000)
                            reliable(yes)
                            dir("/tmp/disk-buffer")
                     )
              );
 };
```

In the following case normal disk-buffer() is used.

```
destination d_demo {
    network(
            "127.0.0.1"
            port(3333)
            disk-buffer(
                mem-buf-length(10000)
                disk-buf-size(2000000)
```

```
                reliable(no)
                dir("/tmp/disk-buffer")
            )
        );
};
```

flush_limit()

Type:	number
Default:	5000

Description: The number of messages that syslog-ng PE sends to the Elasticsearch server in a single batch.

- If *flush_limit* is set to 1: syslog-ng PE sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, syslog-ng PE repeats sending the message, as set in the *retries()* parameter. If sending the message fails for *retries()* times, syslog-ng PE drops the message.

  This method ensures reliable message transfer, but is slow.
- If *flush_limit* is higher than 1: syslog-ng PE sends messages in a batch, and receives the response asynchronously. In case of a problem, syslog-ng PE cannot resend the messages.

  This method is relatively fast (depending on the size of *flush_limit*), but the transfer is not reliable. In transport mode, several messages can be lost before syslog-ng PE recognizes the error. Node mode is more reliable in this sense, because the message loss rate is significantly lower.
- If *concurrent-requests* is higher than 1, syslog-ng PE can send multiple batches simultaneously, increasing performance (and also the number of messages that can be lost in case of an error). For details, see the section called “concurrent_requests()”.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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.

index()

Type:	string
Default:	N/A

Description: Name of the Elasticsearch index to store the log messages. You can use macros and templates as well. For example, **`index("syslog-ng_ ${YEAR}.${MONTH}.${DAY}")`**.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

on-error()

Accepted values:	drop-message|drop-property|fallback-to-string|silently-drop-message|silently-drop-property|silently-fallback-to-string
Default:	Use the global setting (which defaults to *drop-message*)

Description: Controls what happens when type-casting fails and syslog-ng PE cannot convert some data to the specified type. By default, syslog-ng PE drops the entire message and logs the error. Currently the *value-pairs()* option uses the settings of *on-error()*.

- *drop-message*: Drop the entire message and log an error message to the *internal()* source. This is the default behavior of syslog-ng PE.
- *drop-property*: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the *internal()* source.
- *fallback-to-string*: Convert the property to string and log an error message to the *internal()* source.
- *silently-drop-message*: Drop the entire message silently, without logging the error.

- *silently-drop-property*: Omit the affected property (macro, template, or message-field) silently, without logging the error.

  silently-fallback-to-string: Convert the property to string silently, without logging the error.

port()

Type:	number
Default:	9300

Description: The port number of the Elasticsearch server. This option is used only in transport mode: **client_mode("transport")**

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng PE attempts to send a message to this destination. If syslog-ng PE could not send a message, it will try again until the number of attempts reaches *retries*, then drops the message.

resource()

Type:	string
Default:	N/A

Description: The list of Elasticsearch resources to load, separated by semicolons. For example, **resource ("/home/user/elasticsearch/elasticsearch.yml;/home/user/elasticsearch/elasticsearch2.yml").**

server()

Type:	list of hostnames
Default:	127.0.0.1

Description: Specifies the hostname or IP address of the Elasticsearch server. When specifying an IP address, IPv4 (for example, **192.168.0.1**) or IPv6 (for example, **[::1]**) can be used as well. When specifying multiple addresses, use space to separate the

addresses, for example, **server("127.0.0.1 remote-server-hostname1 remote-server-hostname2")**

This option is used only in transport mode: **client_mode("transport")**

template()

Type:	template or template function
Default:	$(format-json --scope rfc5424 --exclude DATE --key ISODATE @timestamp-p=${ISODATE})

Description: The message as sent to the Elasticsearch server. Typically, you will want to use the command-line notation of the **format-json** template function.

To add a **@timestamp** field to the message, for example, to use with Kibana, include the **@timestamp=${ISODATE}** expression in the template. For example: **template($(format-json --scope rfc5424 --exclude DATE --key ISODATE @timestamp=${ISODATE}))**

For details on formatting messages in JSON format, see the section called “format-json”.

throttle()

Type:	number (messages per second)
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:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called “Date-related macros”.

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global *ts-format()* parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

type()

Type:	string
Default:	N/A

Description: The type of the index. For example, **type("test")**.

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 example, syslog-ng PE can store the messages of client hosts in a separate file for each host. For more information on available macros see the section called "Macros of syslog-ng PE".

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 the section called "file() destination options".

Declaration:

```
file(filename options());
```

Example 7.8. Using the file() driver

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

Example 7.9. Using the file() driver with macros in the file name and a template for the message

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

NOTE:

When using this destination, update the configuration of your log rotation program to rotate these files. Otherwise, the log files can become very large.

Also, after rotating the log files, reload syslog-ng PE using the **syslog-ng-ctl reload** command, or use another method to send a SIGHUP to syslog-ng PE.

CAUTION:

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.

file() destination options

The *file()* driver outputs messages to the specified text file, or to a set of files. The *file()* destination has the following options:

CAUTION:

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* command-line parameter of syslog-ng or the global ulimit parameter of your host. For setting the *--fd-limit* command-line parameter of syslog-ng see the syslog-ng(8) manual page. For setting the *ulimit* parameter of the host, see the documentation of your operating system.

create-dirs()

Type:	yes or no
Default:	no

Description: Enable creating non-existing directories.

dir-group()

Type:	string
Default:	Use the global settings

Description: The group of the directories created by syslog-ng. To preserve the original properties of an existing directory, use the option without specifying an attribute: *dir-group()*.

dir-owner()

Type:	string
Default:	Use the global settings

Description: The owner of the directories created by syslog-ng. To preserve the original properties of an existing directory, use the option without specifying an attribute: *dir-owner()*.

dir-perm()

Type:	number (octal notation)
Default:	Use the global settings

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 also the *create-dirs()* option). For octal numbers prefix the number with **0**, for example use **0755** for **rwxr-xr-x**.

To preserve the original properties of an existing directory, use the option without specifying an attribute: *dir-perm()*. Note that when creating a new directory without specifying attributes for *dir-perm()*, the default permission of the directories is masked with the umask of the parent process (typically **0022**).

flags()

Type:	no-multi-line, syslog-protocol, threaded
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. 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, and that the *syslog* driver automatically adds the frame header to the messages.

threaded: The `threaded` flag enables multithreading for the destination. For details on multithreading, see Chapter 18, *Multithreading and scaling in syslog-ng PE*.

NOTE:

The `file` destination uses multiple threads only if the destination filename contains macros.

flush-lines()

Type:	number (messages)
Default:	Use global setting.

Description: Specifies how many lines are sent to a destination at a time. The syslog-ng PE application 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 destination, but also increases message latency.

For optimal performance when sending messages to an syslog-ng PE server, make sure that the `flush-lines()` is smaller than the window size set using the `log-iw-size()` option in the source of your server.

flush-timeout() (OBSOLETE)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

NOTE:

This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the `flush-timeout()` option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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:	Use the global settings

Description: Set the group of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: *group()*.

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 in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (**0**), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- **internal**: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:
- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*
- **dst-idle**: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.
- **host-idle**: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.
- **periodical**: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.

MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **none**: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.
- **global**: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

overwrite-if-older()

Type:	number (seconds)
Default:	0

Description: If set to a value higher than 0, syslog-ng PE 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:	Use the global settings

Description: Set the owner of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: *owner()*.

pad-size()

Type:	number (bytes)
Default:	0

Description: If set, syslog-ng PE will pad output messages to the specified size (in bytes). Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes).

CAUTION:

Hazard of data loss! If the size of the incoming message is larger than the previously set pad-size() value, syslog-ng will truncate the message to the specified size. Therefore, all message content above that size will be lost.

perm()

Type:	number (octal notation)
Default:	Use the global settings

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`.

To preserve the original properties of an existing file, use the option without specifying an attribute: *perm()*.

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 the section called "Macros of syslog-ng PE". 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 for the `'`, `"`, and backspace characters 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.

time-zone()

Type:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global *ts-format()* parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

Storing messages in encrypted files

The syslog-ng PE application can store log messages securely in encrypted, compressed and timestamped binary files. Timestamps can be requested from an external Timestamping Authority (TSA).

Logstore files consist of individual chunks, every chunk can be encrypted, compressed, and timestamped separately. Chunks contain compressed log messages and header information needed for retrieving messages from the logstore file.

The syslog-ng PE application generates an SHA-1 hash for every chunk to verify the integrity of the chunk. The hashes of the chunks are chained together to prevent injecting chunks into the logstore file. The syslog-ng PE application can encrypt the logstore using various algorithms, using the **aes128** encryption algorithm in CBC mode and the **hmac-sha1** hashing (HMAC) algorithm as default. For other algorithms, see the section called "cipher ()" and the section called "digest()".

The destination filename may include macros which get expanded when the message is written, thus a simple *logstore()* driver may create several files. For more information on available macros see the section called "Macros of syslog-ng PE".

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 *logstore()* has a single required parameter that specifies the filename that stores the log messages. For the list of available optional parameters, see the section called "logstore () destination options".

CAUTION:

Hazard of data loss! If your log files are on an NFS-mounted network file system, see the section called "NFS file system for log files".

Declaration:

```
logstore(filename options());
```

Example 7.15. Using the logstore() driver

A simple example saving and compressing log messages.

```
destination d_logstore { logstore("/var/log/messages.lgs" compress(5) ); };
```

A more detailed example that encrypts messages, modifies the parameters for closing chunks, and sets file privileges.

```
destination d_logstore { logstore("/var/log/messages-logstore.lgs"
 encrypt-certificate("/opt/syslog-ng/etc/syslog-ng/keys/10-100-20-40/public-
certificate-of-the-server.pem")
 owner("balabit")
 group("balabit")
 perm(0777)
 ); };
```

The URL to the Timestamping Authority and if needed, the OID of the timestamping policy can be set as global options, or also per logstore destination. The following example specifies the URL and the OID as global options:

```
options {
        timestamp-url("http://10.50.50.50:8080/");
        timestamp-policy("0.4.0.2023.1.1");
};
```

NOTE:

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

CAUTION:

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.

Displaying the contents of logstore files

To display the contents of a logstore file, use the **lgstool** (formerly called **logcat**) command supplied with syslog-ng, for example **lgstool cat /var/log/messages.lgs**. Log messages available in the journal file of the logstore (but not yet written to the logstore file itself) are displayed as well.

To display the contents of encrypted log files, specify the private key of the certificate used to encrypt the file, for example **lgstool cat -k private.key /var/log/messages.lgs**. The contents of the file are sent to the standard output, so it is possible to use **grep** and other tools to find particular log messages, for example **lgstool cat /var/log/messages.lgs |grep 192.168.1.1**. For further details, see lgstool(1).

TIP:

The **lgstool** utility is available for Microsoft Windows operating systems at the syslog-ng Downloads page.

CAUTION:

For files that are in use by syslog-ng, the last chunk that is open cannot be read.

Journal files

Starting with syslog-ng Premium Edition 3.2, syslog-ng PE processes log messages into a journal file before writing them to the logstore file. That way logstore files are consistent even if syslog-ng PE crashes unexpectedly, avoiding losing messages. Note that this does not protect against losing messages if the operating system crashes.

A journal file is automatically created for every logstore file that syslog-ng PE opens. A journal file consists of journal blocks which store the log messages. When a journal block fills up with messages, syslog-ng PE writes the entire block into the logstore file and starts to reuse the journal block (one journal block becomes one chunk in the logstore file). If the messages cannot be written to the logstore file (for example, because the disk becomes unaccessible, or file operations are slow), messages are put to the next journal block (syslog-ng PE uses four blocks by default). When all journal blocks become full, syslog-ng PE will stop processing incoming traffic. syslog-ng PE starts accepting messages to the logstore file again when the first journal block is successfully written to the logstore file. If syslog-ng PE receives a HUP or STOP signal, or no new message arrives into the logstore for the period set in the *time-reap()* parameter, it writes every journal block to the logstore.

When syslog-ng PE is restarted, it automatically processes the journal files to the logstore files, unless a particular logstore file is not part of configuration of syslog-ng PE. Such orphaned journal files can be processed with the **lgstool recover** command. For details on processing orphaned journal files, see the section called “The recover command”.

CAUTION:

- **If a particular logstore destination receives messages at a constant but very low message rate (for example, a 100-byte message every 30 seconds), messages do not get written to the logstore file for a long time, because the journal block does not get full, and messages are more frequent than the *time-reap()* time. This becomes a problem when using logrotate to rotate the logstore files, because log messages will not be in the files they are expected. To avoid this**

situation, either use time-based macros in the filenames of the logstore files, or send a HUP signal to syslog-ng PE right before rotating the logstore files.

- **When every block of a journal becomes full and syslog-ng PE stops processing incoming traffic, *it will not read new messages at all* until a block is successfully written to the related logstore file. This is in contrast with flow-control, where only messages from the source related to the particular destination are not processed.**
- **The messages in the journal file are in plain-text format: they are neither encrypted nor compressed. The journal file has the same permission as the logstore, by default, `root` privileges are required to access them. Make sure you consider this if you change the permissions of the journal file (owner, group, perm) in the syslog-ng PE configuration file.**

NOTE:

Journal files are located in the same folder as the logstore file. The name of the journal file is the same as the logstore file with `.jor` suffix added. For example, the journal file for `messages.lgs` is `messages.lgs.jor`.

The syslog-ng PE application uses a separate journal file for every logstore file. Every journal file is processed by a separate thread. The journal files are mapped into the memory. The journal of an individual logstore file uses up to **`journal-block-size()*journal-block-count()`** memory address, which is 4MB by default. However, if you have several logstore files open in parallel (for example, you are collecting log messages from 500 hosts and storing them in separate files for every host, and the hosts are continuously sending messages) the memory requirements for journaling rise quickly (to ~2GB for the 500 hosts). To limit the memory use of journals, adjust the *`logstore-journal-shmem-threshold()`* global option (by default, it is 512 MB).

If the memory required for the journal files exceeds the *`logstore-journal-shmem-threshold()`* limit, syslog-ng PE will store only a single journal block of every journal file in the memory, and — if more blocks are needed for a journal — store the additional blocks on the hard disk. Opening new logstore files means allocating memory for one new journal block for every new file. In extreme situations involving large traffic, this can lead to syslog-ng PE consuming the entire memory of the system. Adjust the *`journal-block-size()`* and your file-naming conventions as needed to avoid such situations. For details on logstore journals, see the section called “Journal files”.

CAUTION:

If you have a large amount of open logstore files in parrallel (for example, you are using the *`${HOST}`* or *`${PROGRAM}`* macros in your filenames) consider lowering the *`journal-block-size()`* to avoid syslog-ng PE consuming the entire memory of the system.

Example 7.16. Calculating memory usage of logstore journals

If you are using the default settings (4 journal blocks for every logstore journal, one block is 1MB, *logstore-journal-shmem-threshold()* is 512MB), this means that syslog-ng PE will allocate 4MB memory for every open logstore file, up to 512MB if you have 128 open logstore files. Opening a new logstore file would require 4 more megabytes of memory for journaling, bringing the total required memory to 516MB, which is above the *logstore-journal-shmem-threshold()*. In this case, syslog-ng PE switches to storing only a single journal block in the memory, lowering the memory requirements of journaling to 129MB. However, opening more and more logstore files will require more and more memory, and this is not limited, except when syslog-ng PE reaches the maximum number of files that can be open (as set in the *--fd-limit* command-line option).

logstore() destination options

The logstore driver stores log messages in binary files that can be encrypted, compressed, checked for integrity, and timestamped by an external Timestamping Authority (TSA). Otherwise, it is very similar to the *file()* destination.

CAUTION:

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* command-line parameter of syslog-ng or the global ulimit parameter of your host. For setting the *--fd-limit* command-line parameter of syslog-ng see the syslog-ng(8) manual page. For setting the *ulimit* parameter of the host, see the documentation of your operating system.

NOTE:

When using this destination, update the configuration of your log rotation program to rotate these files. Otherwise, the log files can become very large.

Also, after rotating the log files, reload syslog-ng PE using the **syslog-ng-ctl reload** command, or use another method to send a SIGHUP to syslog-ng PE.

The *logstore()* has a single required parameter that specifies the filename that stores the log messages.

Declaration:

```
logstore(filename options());
```

The *logstore()* destination has the following options:

cipher()

Type:	string
Default:	**aes-128-cbc**

Description: Set the cipher method used to encrypt the logstore. The following cipher methods are available: **aes-128-cbc**, **aes-128-cfb**, **aes-128-cfb1**, **aes-128-cfb8**, **aes-128-ecb**, **aes-128-ofb** , **aes-192-cbc**, **aes-192-cfb**, **aes-192-cfb1**, **aes-192-cfb8**, **aes-192-ecb**, **aes-192-ofb** , **aes-256-cbc**, **aes-256-cfb**, **aes-256-cfb1**, **aes-256-cfb8**, **aes-256-ecb**, **aes-256-ofb** , **aes128** , **aes192** , **aes256**, **bf** , **bf-cbc** , **bf-cfb**, **bf-ecb** , **bf-ofb** , **blowfish**, **cast** , **cast-cbc** , **cast5-cbc** , **cast5-cfb**, **cast5-ecb**, **cast5-ofb** , **des**, **des-cbc**, **des-cfb** , **des-cfb1** , **des-cfb8** , **des-ecb** , **des-ede**, **des-ede-cbc**, **des-ede-cfb** , **des-ede-ofb**, **des-ede3** , **des-ede3-cbc**, **des-ede3-cfb**, **des-ede3-ofb**, **des-ofb** , **des3** , **desx** , **desx-cbc**, **rc2**, **rc2-40-cbc** , **rc2-64-cbc**, **rc2-cbc**, **rc2-cfb**, **rc2-ecb** , **rc2-ofb**, **rc4**, and **rc4-40**. By default, syslog-ng PE uses the **aes-128-cbc** method.

Note that the size of the digest hash must be equal to or larger than the key size of the cipher method. For example, to use the **aes-256-cbc** cipher method, the digest method must be at least **SHA-256**. This option is available in syslog-ng PE 3.1 and later.

chunk-size()

Type:	number (kilobytes)
Default:	128

Description: This option is obsolete. Use the *journal-block-size()* option instead.

Size of a logstore chunk in kilobytes. Note that this size refers to the compressed size of the chunk. Also, the gzip library used for compressing the messages has a 32k long buffer, so messages may not appear in the actual logfile until this buffer is not filled. Logstore chunks are closed when they reach the specified size, or when the time limit set in *chunk-time()* expires.

chunk-time()

Type:	number (seconds)
Default:	5

Description: This option is obsolete.

Time limit in seconds: syslog-ng PE closes the chunk if no new messages arrive until the time limit expires. Logstore chunks are closed when the time limit expires, or when they reach the size specified in the *chunk-size()* parameter. If the time limit set in the *time-reap()* parameter expires, the entire file is closed.

compress()

Type:	number (between 0-9)
Default:	3

Description: Compression level. `0` means uncompressed files, while 1-9 is the compression level used by gzip (`9` means the highest but slowest compression, `3` is usually a good compromise).

create-dirs()

Type:	yes or no
Default:	no

Description: Enable creating non-existing directories.

digest()

Type:	string
Default:	SHA1

Description: Set the digest method to use. The following digest methods are available: **`MD4`**, **`MD5`**, **`SHA0 (SHA)`**, **`SHA1`**, **`RIPEMD160`**, **`SHA224`**, **`SHA256`**, **`SHA384`**, and **`SHA512`**. By default, syslog-ng PE uses the **`SHA1`** method.

Note that the size of the digest hash must be equal to or larger than the key size of the cipher method. For example, to use the **`aes-256-cbc`** cipher method, the digest method must be at least **`SHA256`**. This option is available in syslog-ng PE 3.1 and later.

dir-group()

Type:	string
Default:	Use the global settings

Description: The group of the directories created by syslog-ng. To preserve the original properties of an existing directory, use the option without specifying an attribute: *`dir-group()`*.

dir-owner()

Type:	string
Default:	Use the global settings

Description: The owner of the directories created by syslog-ng. To preserve the original properties of an existing directory, use the option without specifying an attribute: *dir-owner()*.

dir-perm()

Type:	number (octal notation)
Default:	Use the global settings

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 also the *create-dirs()* option). For octal numbers prefix the number with **0**, for example use **0755** for **rwxr-xr-x**.

To preserve the original properties of an existing directory, use the option without specifying an attribute: *dir-perm()*. Note that when creating a new directory without specifying attributes for *dir-perm()*, the default permission of the directories is masked with the umask of the parent process (typically **0022**).

encrypt-certificate()

Type:	filename
Default:	none

Description: Name of a file, that contains an X.509 certificate (and the public key) in PEM format. The syslog-ng PE application uses this certificate to encrypt the logstore files which can be decrypted using the private key of the certificate.

flags()

Type:	serialized
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- The *serialized* flag instructs the driver to store the log messages in a serialized format. When using the **lgstool** utility to display messages from the logstore, the messages can be reformatted with a template only if the *serialized* flag has been enabled on the logstore.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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:	Use the global settings

Description: Set the group of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: *group()*.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

journal-block-count()

Type:	number (1-255)
Default:	4

Description: The number of blocks in the journal file. If set to **0**, syslog-ng will set it to the default value (**4**). The maximal value is **255**. If *journal-block-count()* is set higher than 255, syslog-ng will use the maximum value.

NOTE:

By default, journal files are mapped into the memory of the host. To influence the amount of memory addresses used by journal files, see the *logstore-journal-shmem-threshold()* global option.

Example 7.17. Setting journal block number and size

The following example sets the size of a journal block to 512KB and increases the number of blocks to 5.

```
destination  d_logstore  {
        logstore("/var/log/messages-logstore.lgs"
             encrypt-certificate  ("/opt/syslog-ng/etc/syslog-ng/keys/public-
server-certificate.pem")
             journal-block-size(524288)
             journal-block-count(5));
 };
```

journal-block-size()

Type:	number (bytes)
Default:	1048576

Description: The size of blocks (in bytes) in the journal file. The size of the block must be a multiple of the page size: if not, syslog-ng PE automatically increases it to the next multiple of the page size. The maximum size of a journal block is 32MB, the minimum size is 256KB. If the value specified as *journal-block-size()* is lower than minimum size or higher than the maximum size, syslog-ng PE will use the minimum or maximum size, respectively.

NOTE:

- At least one journal block for every logstore file open is mapped into the memory. For details on logstore journals, see the section called "Journal files".
- The size of the journal block is not equal with the size of logstore chunks, because the records in the logstore file can be encrypted or compressed.

Example 7.18. Setting journal block number and size

The following example sets the size of a journal block to 512KB and increases the number of blocks to 5.

```
destination  d_logstore  {
        logstore("/var/log/messages-logstore.lgs"
             encrypt-certificate  ("/opt/syslog-ng/etc/syslog-ng/keys/public-
server-certificate.pem")
             journal-block-size(524288)
             journal-block-count(5));
 };
```

owner()

Type:	string
Default:	Use the global settings

Description: Set the owner of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: *owner()*.

perm()

Type:	number (octal notation)
Default:	Use the global settings

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`**.

To preserve the original properties of an existing file, use the option without specifying an attribute: *perm()*.

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 the section called "Macros of syslog-ng PE". 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.

throttle()

Type:	number (messages per second)
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.

timestamp-freq()

Type:	number (seconds)
Default:	Use global setting.

Description: The minimum time (in seconds) that should expire between two timestamping requests. When syslog-ng closes a chunk, it checks how much time has expired since the last timestamping request: if it is higher than the value set in the *timestamp-freq()* parameter, it requests a new timestamp from the authority set in the *timestamp-url()* parameter.

By default, timestamping is disabled: the *timestamp-freq()* global option is set to 0. To enable timestamping, set it to a positive value.

timestamp-policy()

Type:	string
Default:	

Description: If the Timestamping Server has timestamping policies configured, specify the OID of the policy to use with this parameter. syslog-ng PE will include this ID in the timestamping requests sent to the TSA. This option is available in syslog-ng PE 3.1 and later.

timestamp-url()

Type:	string
Default:	Use global setting.

Description: The URL of the Timestamping Authority used to request timestamps to sign logstore chunks. Note that syslog-ng PE currently supports only Timestamping Authorities that conform to *RFC3161 Internet X.509 Public Key Infrastructure Time-Stamp Protocol*, other protocols like *Microsoft Authenticode Timestamping* are not supported.

time-zone()

Type:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called “Date-related macros”.

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global *ts-format()* parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

Storing messages in a MongoDB database

The *mongodb()* driver sends messages to a MongoDB database. MongoDB is a schema-free, document-oriented database. For the list of available optional parameters, see the section called "mongodb() destination options".

> NOTE:
>
> In order to use this destination, syslog-ng Premium Edition must run in server mode. Typically, only the central syslog-ng Premium Edition server uses this destination. For details on the server mode, see the section called "Server mode".

Declaration:

```
mongodb(parameters);
```

The *mongodb()* driver does not support creating indexes, as that can be a very complex operation in MongoDB. If needed, the administrator of the MongoDB database must ensure that indexes are created on the collections.

The *mongodb()* driver does not add the **_id** field to the message: the MongoDB server will do that automatically, if none is present. If you want to override this field from syslog-ng PE, use the *key()* parameter of the *value-pairs()* option.

The syslog-ng PE *mongodb()* driver is compatible with MongoDB server version 1.4 and newer.

> NOTE:
>
> By default, syslog-ng PE handles every message field as a string. For details on how to send selected fields as other types of data (for example, handle the PID as a number), see the section called "Specifying data types in value-pairs".

Example 7.19. Using the mongodb() driver

The following example creates a *mongodb()* destination using only default values.

```
destination d_mongodb {
      mongodb();
};
```

The following example displays the default values, and is equivalent with the previous example.

```
destination  d_mongodb  {
       mongodb(
              servers("localhost:27017")
              database("syslog")
              collection("messages")
              value-pairs(
                     scope("selected-macros"  "nv-pairs"  "sdata")
              )
       );
};
```

Procedure 7.7. How syslog-ng PE connects the MongoDB server

When syslog-ng PE connects the MongoDB server during startup, it completes the following steps.

1. The syslog-ng PE application connects the first address listed in the *servers()* option.
2. - If the server is accessible and it is a master MongoDB server, syslog-ng PE authenticates on the server (if needed), then starts sending the log messages to the server.
   - If the server is not accessible, or it is not a master server in a MongoDB replicaset and it does not send the address of the master server, syslog-ng PE connects the next address listed in the *servers()* option.
   - If the server is not a master server in a MongoDB replicaset, but it sends the address of the master server, syslog-ng PE connects the received address.
3. When syslog-ng PE connects the master MongoDB server, it retrieves the list of replicas (from the **replSet** option of the server), and appends this list to the *servers()* option.

   CAUTION:

   - **This means that syslog-ng PE can send log messages to addresses that are not listed in its configuration.**
   - **Make sure to include the address of your master server in your syslog-ng PE configuration file, otherwise you risk losing log messages if all the addresses listed in the syslog-ng PE configuration are offline.**
   - **Addresses retrieved from the MongoDB servers are not stored, and can be lost when syslog-ng PE is restarted. The retrieved addresses are not lost if the *server()* option of the destination was not changed in the configuration file since the last restart.**

- **The failover mechanism used in the *mongodb()* driver is different from the client-side failover used in other drivers.**

4. The syslog-ng PE application attempts to connect another server if the *servers()* list contains at least two addresses, and one of the following events happens:
   - The **safe-mode()** option is set to **no**, and the MongoDB server becomes unreachable.
   - The **safe-mode()** option is set to **yes**, and syslog-ng PE cannot insert a log message into the database because of an error.

   In such case, syslog-ng PE starts to connect the addresses in from the *servers()* list (starting from the first address) to find the new master server, authenticates on the new server (if needed), then continues to send the log messages to the new master server.

   During this failover step, one message can be lost if the *safe-mode()* option is disabled.
5. If the original master becomes accessible again, syslog-ng PE will automatically connect to the original master.

mongodb() destination options

The *mongodb()* driver sends messages to a MongoDB database. MongoDB is a schema-free, document-oriented database.

NOTE:

In order to use this destination, syslog-ng Premium Edition must run in server mode. Typically, only the central syslog-ng Premium Edition server uses this destination. For details on the server mode, see the section called “Server mode”.

The *mongodb()* destination has the following options:

collection()

Type:	string
Default:	messages

Description: The name of the MongoDB collection where the log messages are stored (collections are similar to SQL tables). Note that the name of the collection must not start with a dollar sign ($), and that it may contain dot (.) characters.

CAUTION:

Hazard of data loss! The syslog-ng PE application does not verify that the specified collection name does not contain invalid characters. If you

specify a collection with an invalid name, the log messages sent to the MongoDB database will be irrevocably lost without any warning.

database()

Type:	string
Default:	syslog

Description: The name of the MongoDB database where the log messages are stored. Note that the name of the database must not start with a dollar sign ($) and it cannot contain dot (.) characters.

CAUTION:

Hazard of data loss! The syslog-ng PE application does not verify that the specified database name does not contain invalid characters. If you specify a database with an invalid name, the log messages sent to the MongoDB database will be irrevocably lost without any warning.

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()

Type:	yes|no
Default:	no
	Description: If set to `yes`, syslog-ng PE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of *`reliable()`* option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored. This option has priority over `--qdisk-dir=`.

CAUTION:

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file, or at least remove the relevant persist-entry.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file or the relevant entry is not deleted after modifying the `dir()` option, then following a restart, syslog-ng PE will look for or create disk-buffer files in their old location. To ensure that syslog-ng PE uses the new `dir()` setting, the persist file must not contain any information about the destinations which the disk-buffer file in question belongs to.

disk-buf-size()

Type:	number (bytes)
Default:	

Description: This is a required option. The maximum size of the disk-buffer in bytes. The minimum value is 1048576 bytes. If you set a smaller value, the minimum value will be used automatically. It replaces the old `log-disk-fifo-size()` option.

mem-buf-length()

Type:	number (messages)
Default:	10000

Description: Use this option if the option `reliable()` is set to **no**. This option contains the number of messages stored in overflow queue. It replaces the old `log-fifo-size()` option. It inherits the value of the global `log-fifo-size()` option if provided. If it is not provided, the default value is **10000** messages. Note that this option will be ignored if the option `reliable()` is set to **yes**.

mem-buf-size()	
Type:	number (bytes)
Default:	163840000
	Description: Use this option if the option *reliable()* is set to **yes**. This option contains the size of the messages in bytes that is used in the memory part of the disk buffer. It replaces the old *log-fifo-size()* option. It does not inherit the value of the global *log-fifo-size()* option, even if it is provided. Note that this option will be ignored if the option *reliable()* is set to **no**.

quot-size()	
Type:	number (messages)
Default:	64
	Description: The number of messages stored in the output buffer of the destination.

Options *reliable()* and *disk-buf-size()* are required options.

Example 7.20. Examples for using disk-buffer()

In the following case reliable disk-buffer() is used.

```
destination  d_demo  {
       network(
                    "127.0.0.1"
                    port(3333)
                    disk-buffer(
                           mem-buf-size(10000)
                           disk-buf-size(2000000)
                           reliable(yes)
                           dir("/tmp/disk-buffer")
                    )
             );
};
```

In the following case normal disk-buffer() is used.

```
destination  d_demo  {
       network(
                    "127.0.0.1"
                    port(3333)
                    disk-buffer(
                          mem-buf-length(10000)
                          disk-buf-size(2000000)
                          reliable(no)
                          dir("/tmp/disk-buffer")
                    )
             );
 };
```

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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.

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 in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

on-error()

Accepted values:	drop-message|drop-property|fallback-to-string|silently-drop-message|silently-drop-property|silently-fallback-to-string
Default:	Use the global setting (which defaults to *drop-message*)

Description: Controls what happens when type-casting fails and syslog-ng PE cannot convert some data to the specified type. By default, syslog-ng PE drops the entire message and logs the error. Currently the *value-pairs()* option uses the settings of *on-error()*.

- *drop-message*: Drop the entire message and log an error message to the *internal()* source. This is the default behavior of syslog-ng PE.
- *drop-property*: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the *internal()* source.
- *fallback-to-string*: Convert the property to string and log an error message to the *internal()* source.
- *silently-drop-message*: Drop the entire message silently, without logging the error.
- *silently-drop-property*: Omit the affected property (macro, template, or message-field) silently, without logging the error.
- *silently-fallback-to-string*: Convert the property to string silently, without logging the error.

password()

Type:	string
Default:	n/a

Description: Password of the database user.

path()

Type:	string
Default:	empty

Description: If the *path()* option is set, syslog-ng PE will connect to the database using the specified UNIX domain socket. Note that you cannot set the *path()* and the *servers()* options at the same time.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng PE attempts to send a message to this destination. If syslog-ng PE could not send a message, it will try again until the number of attempts reaches *retries*, then drops the message.

For MongoDB operations, syslog-ng PE uses a one-minute timeout: if an operation times out, syslog-ng PE assumes the operation has failed.

safe-mode()

Type:	yes or no
Default:	yes

Description: If *safe-mode()* is enabled, syslog-ng PE performs an extra check after each insert to verify that the insert succeeded. The insert is successful only if this second check is successful.

Note that disabling this option increases the performance of the driver, but can result in message loss. Using **safe-mode(yes)** is technically equivalent of using the RLTP™ protocol between syslog-ng PE and the MongoDB server. If you use the **reliable(yes)** option of *disk-buffer()* in your destinations, make sure that the *safe-mode()* option of the *mongodb()* destination is set to **yes**.

servers()

Type:	list of hostname:port pairs
Default:	127.0.0.1:27017

Description: Specifies the hostname or IP address and the port number of the database server. When specifying an IP address, IPv4 (for example, **192.168.0.1**) or IPv6 (for example, **[::1]**) can be used as well.

To send the messages to a MongoDB replicaset, specify the addresses of the database servers as a comma-separated list, for example: **servers (192.168.1.1:27017,192.168.3.3:27017)**

For details on how syslog-ng PE connects the MongoDB server, see Procedure 7.7, “How syslog-ng PE connects the MongoDB server”.

To connect to the server using a UNIX domain socket, use path option. Note that you cannot set the *path()* and the *servers()* options at the same time.

time-zone()

Type:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

username()

Type:	string
Default:	n/a

Description: Name of the database user. Note that the *mongodb()* driver currently does not support TLS-encrypted authentication.

value-pairs()

Type:	parameter list of the *value-pairs()* option
Default:	`scope("selected-macros" "nv-pairs")`

Description: The *value-pairs()* option creates structured name-value pairs from the data and metadata of the log message. For details on using *value-pairs()*, see the section called "Structuring macros, metadata, and other value-pairs".

> NOTE:
>
> Empty keys are not logged.

Sending messages to a remote log server using the RFC3164 protocol (network() driver)

The network() destination driver can send syslog messages conforming to RFC3164 from the network using the TCP, TLS, and UDP networking protocols.

You can use the RLTP™ protocol as well. For details about the RLTP™ protocol, see Chapter 12, *Reliable Log Transfer Protocol™* .

- UDP is a simple datagram oriented protocol, which provides "best effort service" to transfer messages between hosts. It may lose messages, and no attempt is made to retransmit lost messages. The *BSD-syslog* protocol traditionally uses UDP.

  Use UDP only if you have no other choice.

  For details on minimizing message loss when using UDP, see *Collecting log messages from UDP sources*.
- TCP provides connection-oriented service: the client and the server establish a connection, each message is acknowledged, and lost packets are resent. TCP can detect lost connections, and messages are lost, only if the TCP connection breaks. When a TCP connection is broken, messages that the client has sent but were not yet received on the server are lost.
- The syslog-ng application supports TLS (Transport Layer Security, also known as SSL) over TCP. For details, see the section called “Encrypting log messages with TLS”.

When you send your log messages from a syslog-ng PE client through the network to a syslog-ng PE server, you can use different protocols and options. Every combination has its advantages and disadvantages. The most important thing is to use matching protocols and options, so the server handles the incoming log messages properly. For details, see the section called “Things to consider when forwarding messages between syslog-ng PE hosts”.

Declaration:

```
network("<destination-address>" [options]);
```

The *network()* destination has a single required parameter that specifies the destination host address where messages should be sent. If name resolution is configured, you can use the hostname of the target server. By default, syslog-ng PE sends messages using the TCP protocol to port 514.

Example 7.21. Using the network() driver

TCP destination that sends messages to **10.1.2.3**, port **1999**:

```
destination d_tcp { network("10.1.2.3" port(1999)); };
```

If name resolution is configured, you can use the hostname of the target server as well.

```
destination d_tcp { network("target_host" port(1999)); };
```

TCP destination that sends messages to the `::1` IPv6 address, port `2222`.

```
destination  d_tcp6  {
       network(
              "::1"
              port(2222)
              transport(tcp)
              ip-protocol(6)
              );
 };
```

To send messages using the IETF-syslog message format without using the IETF-syslog protocol, enable the *syslog-protocol* flag. (For details on how to use the IETF-syslog protocol, see the section called “syslog() destination options”.)

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

network() destination options

The *network()* driver sends messages to a remote host (for example a syslog-ng server or relay) on the local intranet or internet using the RFC3164 syslog protocol (for details about the protocol, see the section called “BSD-syslog or legacy-syslog messages”). The *network()* driver supports sending messages using the UDP, TCP, RLTP ™or the encrypted TLS networking protocols.

These destinations have the following options:

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()

Type:	yes|no

Default:	no

Description: If set to **yes**, syslog-ng PE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to **no**, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of *reliable()* option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored. This option has priority over *--qdisk-dir=*.

CAUTION:

When creating a new *dir()* option for a disk buffer, or modifying an existing one, make sure you delete the persist file, or at least remove the relevant persist-entry.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file or the relevant entry is not deleted after modifying the *dir()* option, then following a restart, syslog-ng PE will look for or create disk-buffer files in their old location. To ensure that syslog-ng PE uses the new *dir()* setting, the persist file must not contain any information about the destinations which the disk-buffer file in question belongs to.

disk-buf-size()

Type:	number (bytes)
Default:	

	Description: This is a required option. The maximum size of the disk-buffer in bytes. The minimum value is 1048576 bytes. If you set a smaller value, the minimum value will be used automatically. It replaces the old *log-disk-fifo-size()* option.
mem-buf-length()	
Type:	number (messages)
Default:	10000
	Description: Use this option if the option *reliable()* is set to **no**. This option contains the number of messages stored in overflow queue. It replaces the old *log-fifo-size()* option. It inherits the value of the global *log-fifo-size()* option if provided. If it is not provided, the default value is **10000** messages. Note that this option will be ignored if the option *reliable()* is set to **yes**.
mem-buf-size()	
Type:	number (bytes)
Default:	163840000
	Description: Use this option if the option *reliable()* is set to **yes**. This option contains the size of the messages in bytes that is used in the memory part of the disk buffer. It replaces the old *log-fifo-size()* option. It does not inherit the value of the global *log-fifo-size()* option, even if it is provided. Note that this option will be ignored if the option *reliable()* is set to **no**.
quot-size()	
Type:	number (messages)
Default:	64
	Description: The number of messages stored in the output buffer of the destination.

Options *reliable()* and *disk-buf-size()* are required options.

Example 7.22. Examples for using disk-buffer()

In the following case reliable disk-buffer() is used.

```
destination  d_demo  {
       network(
                     "127.0.0.1"
                     port(3333)
                     disk-buffer(
                           mem-buf-size(10000)
                           disk-buf-size(2000000)
                           reliable(yes)
                           dir("/tmp/disk-buffer")
                     )
              );
 };
```

In the following case normal disk-buffer() is used.

```
destination  d_demo  {
       network(
                     "127.0.0.1"
                     port(3333)
                     disk-buffer(
                           mem-buf-length(10000)
                           disk-buf-size(2000000)
                           reliable(no)
                           dir("/tmp/disk-buffer")
                     )
              );
 };
```

failover-servers()

Type:	list of IP addresses and fully-qualified domain names
Default:	0

Description: Available only in syslog-ng Premium Edition version 3.2 and later. Specifies a secondary destination server where log messages are sent if the primary server becomes unaccessible. To list several failover servers, separate the address of the servers with comma. The time syslog-ng PE waits for the a server before switching to the next failover server is set in the *time-reopen()* option. For details about how client-side failover works, see the section called “Client-side failover”.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

NOTE:

This option is not available for the connection-less UDP protocol, because in this case the client does not detect that the destination becomes unaccessible.

Example 7.23. Specifying failover servers for syslog() destinations

The following example specifies two failover servers for a simple syslog() destination.

```
destination d_syslog_tcp{
                syslog("10.100.20.40"
                transport("tcp")
                port(6514)
                failover-servers("10.2.3.4", "myfailoverserver")
                );};
```

The following example specifies a failover server for a network() destination that uses TLS encryption.

```
destination d_syslog_tls{
                network("10.100.20.40"
                transport("tls")
                port(6514)
                failover-servers("10.2.3.4", "myfailoverserver")
                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'))
                );};
```

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The `syslog-protocol` flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. 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, and that the *syslog* driver automatically adds the frame header to the messages.

flush-lines()

Type:	number (messages)
Default:	Use global setting.

Description: Specifies how many lines are sent to a destination at a time. The syslog-ng PE application 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 destination, but also increases message latency.

For optimal performance when sending messages to an syslog-ng PE server, make sure that the *flush-lines()* is smaller than the window size set using the *log-iw-size()* option in the source of your server.

flush-timeout() (OBSOLETE)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush-lines()* option.

NOTE:

This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the *flush-timeout()* option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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 (type of service)
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip-ttl()

Type:	number (hops)
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 reloaded. 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.

localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to before connecting to target.

localport()

Type:	number (port number)
Default:	0

Description: The port number to bind to. Messages are sent from this port.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (**0**), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- **internal**: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:
- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*
- **dst-idle**: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **host-idle**: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **periodical**: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **none**: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.

- **global**: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

port() or localport()

Type:	number (port number)
Default:	UDP — 514, TCP — 601, TLS — 6514.

Description: The port number to connect to.

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. For details, see the **socket(7)** manual page.

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. For details, see the **socket(7)** manual page.

so-rcvbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. For details, see the **socket(7)** manual page.

so-sndbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket send buffer in bytes. For details, see the **socket(7)** manual page.

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. This 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.

CAUTION:

When using the *spoof-source* option, syslog-ng PE automatically truncates long messages to 1024 bytes, regardless of the settings of *log-msg-size()*.

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 the section called "Macros of syslog-ng PE". 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.

> NOTE:
>
> If a message uses the IETF-syslog format (RFC5424), only the text of the message can be customized (that is, the $MESSAGE part of the log), the structure of the header is fixed.

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backspace characters 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 (messages per second)
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:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various options related to TLS encryption, for example, key/certificate files and trusted CA locations. TLS can be used only with tcp-based transport protocols. For details, see the section called "TLS options".

transport()

Type:	rltp, udp, tcp, or tls
Default:	tcp

Description: Specifies the protocol used to send messages to the destination server.

If you use the **udp** transport, syslog-ng PE automatically sends multicast packets if a multicast destination address is specified. The **tcp** transport does not support multicasting.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global *ts-format()* parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

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 the section called “pipe() destination options”.

Declaration:

```
pipe(filename);
```

CAUTION:

Starting with syslog-ng PE 3.0.2, pipes are created automatically. In earlier versions, you had to create the pipe using the mkfifo(1) command.

Example 7.25. Using the pipe() driver

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

pipe() destination options

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

The *pipe()* destination has the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. 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, and that the *syslog* driver automatically adds the frame header to the messages.

flush-lines()

Type:	number (messages)
Default:	Use global setting.

Description: Specifies how many lines are sent to a destination at a time. The syslog-ng PE application 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 destination, but also increases message latency.

For optimal performance when sending messages to an syslog-ng PE server, make sure that the *flush-lines()* is smaller than the window size set using the *log-iw-size()* option in the source of your server.

flush-timeout() (OBSOLETE)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush-lines()* option.

NOTE:

This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the *flush-timeout()* option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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:	Use the global settings

Description: Set the group of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: *group()*.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (**0**), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

internal: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:

- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*

dst-idle: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.

- *MARK* signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

host-idle: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.

- *MARK* signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

periodical: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.

- *MARK* signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

none: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.

global: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

owner()

Type:	string
Default:	Use the global settings

Description: Set the owner of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: *owner()*.

pad-size()

Type:	number (bytes)
Default:	0

Description: If set, syslog-ng PE will pad output messages to the specified size (in bytes). Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes).

CAUTION:

Hazard of data loss! If the size of the incoming message is larger than the previously set pad-size() value, syslog-ng will truncate the message to the specified size. Therefore, all message content above that size will be lost.

perm()

Type:	number (octal notation)
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 the section called “Macros of syslog-ng PE”. 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 for the `'`, `"`, and backspace characters 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 (messages per second)
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:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called “Date-related macros”.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see the section called “A note on timezones and timestamps”.

Sending messages to external applications

The *program()* driver starts an external application or script and sends the log messages to its standard input (*stdin*). Usually, every message is a single line (ending with a newline character), which your script can process. Make sure that your script runs in a loop and keeps reading the standard input — it should not exit. (If your script exits, syslog-ng PE tries to restart it.)

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 the section called "program() destination options".

Declaration:

```
program(command_to_run);
```

NOTE:

- The syslog-ng PE application must be able to start and restart the external program, and have the necessary permissions to do so. For example, if your host is running AppArmor, you might have to modify your AppArmor configuration to enable syslog-ng PE to execute external applications.
- The syslog-ng PE application executes program destinations through the standard system shell. If the system shell is not bash and you experience problems with the program destination, try changing the `/bin/sh` link to `/bin/bash`.
- If the external program exits, the syslog-ng PE application automatically restarts it. 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.
- When the syslog-ng PE application stops, it will automatically stop the external program. To avoid restarting the application when syslog-ng PE is only reloaded, enable the *keep-alive()* option in the program destination.
- Certain external applications buffer the log messages, which might cause unexpected latency and other problems. For example, if you send the log messages to an external Perl script, Perl uses a line buffer for terminal output and block buffer otherwise. You might want to disable buffering in the external application.

Example 7.26. Using the program() destination driver

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. Make sure to end your template with a newline character (**\n**).

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

The following shell script writes the incoming messages into the `/tmp/testlog` file.

```
#!/bin/bash
 while  read  line  ;  do
 echo  $line  >>  /tmp/testlog
 done
```

program() destination options

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

The *program()* destination has the following options:

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()

Type:	yes|no
Default:	no
	Description: If set to **yes**, syslog-ng PE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to **no**, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of *reliable()* option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored. This option has priority over *--qdisk-dir=*.

CAUTION:

When creating a new *dir()* option for a disk buffer, or modifying an existing one, make sure you delete the persist file, or at least remove the relevant persist-entry.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file or the relevant entry is not deleted after modifying the *dir()* option, then following a restart, syslog-ng PE will look for or create disk-buffer files in their old location. To ensure that syslog-ng PE uses the new *dir()* setting, the persist file must not contain any information about the destinations which the disk-buffer file in question belongs to.

disk-buf-size()

Type:	number (bytes)
Default:	

Description: This is a required option. The maximum size of the disk-buffer in bytes. The minimum value is 1048576 bytes. If you set a smaller value, the minimum value will be used automatically. It replaces the old *log-disk-fifo-size()* option.

mem-buf-length()

Type:	number (messages)
Default:	10000

Description: Use this option if the option *reliable()* is set to **no**. This option contains the number of messages stored in overflow queue. It replaces the old *log-fifo-size()* option. It inherits the value of the global *log-fifo-size()* option if provided. If it is not provided, the default value is **10000** messages. Note that this option will be ignored if the option *reliable()* is set to **yes**.

mem-buf-size()	
Type:	number (bytes)
Default:	163840000
	Description: Use this option if the option *reliable()* is set to **yes**. This option contains the size of the messages in bytes that is used in the memory part of the disk buffer. It replaces the old *log-fifo-size()* option. It does not inherit the value of the global *log-fifo-size()* option, even if it is provided. Note that this option will be ignored if the option *reliable()* is set to **no**.

quot-size()	
Type:	number (messages)
Default:	64
	Description: The number of messages stored in the output buffer of the destination.

Options *reliable()* and *disk-buf-size()* are required options.

Example 7.27. Examples for using disk-buffer()

In the following case reliable disk-buffer() is used.

```
destination  d_demo  {
       network(
                    "127.0.0.1"
                    port(3333)
                    disk-buffer(
                           mem-buf-size(10000)
                           disk-buf-size(2000000)
                           reliable(yes)
                           dir("/tmp/disk-buffer")
                    )
             );
};
```

In the following case normal disk-buffer() is used.

```
destination  d_demo  {
       network(
                    "127.0.0.1"
                    port(3333)
                    disk-buffer(
                          mem-buf-length(10000)
                          disk-buf-size(2000000)
                          reliable(no)
                          dir("/tmp/disk-buffer")
                    )
             );
};
```

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The `syslog-protocol` flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. 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, and that the `syslog` driver automatically adds the frame header to the messages.

flush-lines()

Type:	number (messages)
Default:	Use global setting.

Description: Specifies how many lines are sent to a destination at a time. The syslog-ng PE application 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 destination, but also increases message latency.

For optimal performance when sending messages to an syslog-ng PE server, make sure that the *flush-lines()* is smaller than the window size set using the *log-iw-size()* option in the source of your server.

flush-timeout() (OBSOLETE)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush-lines()* option.

> NOTE:
>
> This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the *flush-timeout()* option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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 (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

keep-alive()

Type:	yes or no
Default:	no

Description: Specifies whether the external program should be closed when syslog-ng PE is reloaded.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (**0**), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- **internal**: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:
- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*
- **dst-idle**: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **host-idle**: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **periodical**: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **none**: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.

- **global**: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

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 the section called "Macros of syslog-ng PE". 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.
Make sure to end your template with a newline character (**\n**).

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the ', ", and backspace characters 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 (messages per second)
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:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the /usr/share/zoneinfo directory.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

Sending SNMP traps

The *snmp()* driver sends SNMP traps using the Simple Network Management Protocol version 2c or version 3. Incoming log messages can be converted to SNMP traps, as the fields of the SNMP messages can be customized using syslog-ng PE macros.

NOTE:

The *snmp()* driver is available in syslog-ng PE version 4 F1 and later.

NOTE:

The *snmp* destination driver currently supports sending SNMP traps only using the UDP transport protocol.

The *snmp()* driver requires the *host()*, *trap-obj()*, and *snmp-obj()* options to be set, as well as the *engine-id()* option in case the SNMPv3 protocol is used. For the list of available optional parameters, see the section called "snmp() destination options".

Declaration:

```
@module snmp
        destination d_snmp {snmp(host() trap-obj() snmp-obj() ...);};
```

By default, syslog-ng PE does not load the *snmp()* module. If you want to use the *snmp()* destination, include the following line in your syslog-ng PE configuration file before defining the destination to load the *snmp()* module. This line is needed only once, even if you use multiple SNMP destinations. For details on modules, see the section called "Loading modules".

```
@module snmp
```

CAUTION:

If syslog-ng PE cannot resolve the destination hostname during startup, it will try to resolve the hostname again when the next message to be sent as an SNMP trap is received. However, if this name resolution fails, the trap will be dropped.

NOTE:

The *snmp()* destination driver does not generate MARK signals itself, but can receive and forward MARK signals.

Example 7.31. Using the snmp() destination driver

The following example defines an SNMP destination that uses the SNMPv2c protocol.

```
@module  snmp

destination  d_snmpv2c{
       snmp(
              version('v2c')
              host('192.168.1.1')
              trap-obj('.1.3.6.1.6.3.1.1.4.1.0',  'Objectid',
'.1.3.6.1.4.1.18372.3.1.1.1.2.1')
              snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.1.0',  'Octetstring',
'Test  SNMP  trap')
              snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.2.0',  'Octetstring',
'admin')
              snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.3.0',  'Ipaddress',
'192.168.1.1')
              );
};
```

The following example defines an SNMP destination that uses the SNMPv3 protocol and uses macros to fill the values of the SNMP objects.

```
@module  snmp

destination  d_snmpv3{
       snmp(
              version('v3')
              host('192.168.1.1')
              port(162)
              engine-id('0xdeadbeefde')
              auth-username('myusername')
              auth-password('password')
              enc-algorithm('AES')
              enc-password('password')
              trap-obj('.1.3.6.1.6.3.1.1.4.1.0',  'Objectid',
'.1.3.6.1.4.1.18372.3.1.1.1.2.1')
              snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.1',  'Octetstring',
'${MESSAGE}')
              snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.2',  'Octetstring',
'admin')
              snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.3',  'Ipaddress',
'${SOURCEIP}')
              );
};
```

Converting Cisco syslog messages to

"clogMessageGenerated" SNMP traps

Starting with version 4 F1, syslog-ng PE can convert the syslog messages sent by Cisco devices to Cisco-specific SNMP traps defined by the CISCO-SYSLOG-MIB (**enterprises.cisco.ciscoMgmt.ciscoCiscoMIB**) is also supported (such traps are also referred to as **clogMessageGenerated** notifications). That way, the incoming log messages can be forwarded to devices used to process and analyze Cisco-specific SNMP traps. For this to work correctly, the following requirements must be met:

- The snmp module of syslog-ng PE must be loaded, that is, the syslog-ng PE configuration file must include the following line:

  ```
  @module snmp
  ```

- The syslog-ng Source Configuration Library (SCL) must be included in the syslog-ng PE configuration file:

  ```
  @include "scl.conf"
  ```

- The pattern database described in the section called “Parsing Cisco-specific message fields with patterndb” must be used to parse the incoming log messages.

To accomplish this, syslog-ng PE has to use a special pattern database to parse the Cisco-specific syslog messages, because these messages do not comply with the standard syslog formats.

For details on the Cisco-specific SNMP trap format, see CISCO-SYSLOG-MIB on the Cisco website.

Parsing Cisco-specific message fields with patterndb

The *${PROGRAM}* part of the syslog messages sent by Cisco devices contain not only the program name, but other important protocol information part as well. The *${PROGRAM}* of these messages contains the Facility, Severity, and the Mnemonic (the Cisco name) of the message. The following pattern database parses these values and makes them available as the **.cisco.Facility**, **.cisco.Severity**, and **.cisco.MsgName**, respectively. The actual log message is available as **.cisco.MsgText**.

```
<patterndb version="4" pub_date="2011-05-03">  <ruleset name="cisco snmp ruleset1"
xml:id="480de478-d4a6-4a7f-bea4-0c0245d361e3">    <description>Pattern for Cisco
messages having BSD timestamps, for example: Jul 01 2010 00:32:59: %SYS-5-CONFIG_I:
Configured from console by console</description>    <url>http://balabit.com</url>
  <pattern>%@ESTRING:.cisco.Facility:-@@ESTRING:.cisco.Severity:-
@@ANYSTRING:.cisco.MsgName@</pattern>        <rules>          <rule
xml:id="09944c71-95eb-4bc0-8575-936931d85713" provider="balabit" class="system">
       <patterns>              <pattern> @ANYSTRING:.cisco.MsgText@</pattern>
      </patterns>          </rule>        </rules>  </ruleset>  <ruleset name="cisco
snmp ruleset2" xml:id="480de478-d4a6-4a7f-bea4-0c0245d361e3">
```

```
<description>Pattern for Cisco messages having cisco-specific timestamps, for
example: 18: Jan 22 10:45:44.543: %SYS-5-CONFIG_I: Configured from console by
console</description>    <url>http://balabit.com</url>        <rules>          <rule
xml:id="09944c71-95eb-4bc0-8575-936931d85714" provider="balabit" class="system">
       <patterns>              <pattern>%@ESTRING:.cisco.Facility:-
@@ESTRING:.cisco.Severity:-@@ESTRING:.cisco.MsgName::@
@ANYSTRING:.cisco.MsgText@</pattern>            </patterns>         </rule>
</rules>  </ruleset></patterndb>
```

Sending clogMessageGenerated SNMP traps

To send out clogMessageGenerated SNMP traps, use the *cisco_snmp()* destination driver. The *cisco-snmp()* destination is actually a modified version of the *snmp()* destination driver.

> NOTE:
>
> The *cisco-snmp()* driver is actually an element of the syslog-ng Source Configuration Library (SCL), a reusable configuration snippet tailored to handle process accounting logs. For details on using or writing SCLs, see the section called "Reusing configuration blocks".

The *cisco-snmp()* driver has the same requirements and options as the *snmp()* destination driver, but automatically fills the clogMessageGenerated-specific fields with the data received from parsing the Cisco-specific syslog messages using the pattern database. For details on the , see the `<INSTALLDIR>/ share/include/scl/snmp/plugin.conf` file.

Declaration:

```
destination d_cisco_snmp {cisco-snmp(host(<hostname>));};
```

Example 7.32. Defining a Cisco-specific SNMP destination

The following example defines an SNMP destination that sends out clogMessageGenerated messages using the SNMPv3 protocol.

```
destination d_cisco_snmp {cisco-snmp(host("192.168.1.1")
version("v3")
engine-id("'0xdeadbeefde'")
auth-username('myusername')
auth-password('password')
enc-password('password'));};
```

snmp() destination options

This driver sends SNMP traps using the SNMP v2c or v3 protocol.

The *snmp()* destination has the following options:

auth-algorithm()

Type:	SHA|sha
Default:	SHA

Description: The authentication method to use. Lowercase values (for example, **sha**) can be used as well.

This option is used with the SNMPv3 protocol.

auth-password()

Type:	string
Default:	empty string

Description: The password used for authentication. If the *auth-username()* option is set but the *auth-password()* is empty, syslog-ng PE will try to authenticate with an empty password.

This option is used with the SNMPv3 protocol.

auth-username()

Type:	string
Default:	

Description: The username used to authenticate on the SNMP server. If this parameter is set, syslog-ng PE will try to authenticate on the SNMP server.

This option is used with the SNMPv3 protocol.

community()

Type:	string
Default:	public

Description: The community string used for SNMPv2c authentication.

This option is used with the SNMPv2c protocol.

enc-algorithm()

Type:	AES|aes
Default:	AES

Description: The encryption method used to encrypt the SNMP traffic. Lowercase values (for example, **aes**) can be used as well.

This option is used with the SNMPv3 protocol.

enc-password()

Type:	string
Default:	

Description: The password used for the encryption. Encryption is used only if the *enc-password()* is not empty.

This option is used with the SNMPv3 protocol.

engine-id()

Type:	number (hexadecimal number)
Default:	

Description: The engine ID is a hexadecimal number at least 32 digits long, starting with **0x**. For example **0xABABABABABABABABABABABABABABABABABABABAB**.

This option is a required parameter when using the SNMPv3 protocol.

host()

Type:	hostname or IP address
Default:	n/a

Description: Hostname of the SNMP server.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

port()

Type:	number (port number)
Default:	162

Description: The port number to connect to.

snmp-obj()

Type:	<oid_of_the_object>, <type_of_the_object>, <value_of_the_object>
Default:	

Description: The *snmp-obj()* option can be used to create custom SNMP trap elements. To create a trap element, specify the OID, type, and value of the element in the *snmp-obj()* option. To send SNMP traps, at least one *snmp-obj()* option must be defined. The *snmp-obj()* option requires the following parameters. Note that syslog-ng PE does not validate the values of these elements.

- **<oid_of_the_object>**: The object id of the SNMP object, for example, **.1.3.6.1.4.1.18372.3.1.1.1.1.1**.
- **<type_of_the_object>**: The type of the object specified as an ASN.1 primitive. One of: **Integer**, **Timeticks**, **Octetstring**, **Counter32**, **Ipaddress**, **Objectid**. The type names are not case sensitive.
- **<value_of_the_object>**: The value of the object as a string. The macros of syslog-ng PE can be used to set these values, making it possible to transfer the content and other metadata from the the syslog message to the SNMP trap. Note that if the value of an **Integer**, **Counter32** or **Timeticks** object is not a number (for example, is an empty string or other not-number string), syslog-ng PE will automatically replace the value with 0. The values of other types of objects are not validated.

Example 7.33. Defining SNMP objects

The following are SNMP object definitions:

```
snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.3', 'Ipaddress', '192.168.1.1')
```

```
snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.2', 'Octetstring', '${MESSAGE}')
```

time-zone()

Type:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

trap-obj()

Type:	<oid_of_the_object>, "Objectid", <value_of_the_object>
Default:	

Description: The *trap-obj()* is a specialized version of the *snmp-obj()* option that is used to identify the SNMP trap object. The type of the trap object is always **Objectid**. The **<oid_of_the_object>** and the **<value_of_the_object>** parameters are identical to the respective parameters of the *snmp-obj()* option. For details on these parameters, see the section called "snmp-obj()".

NOTE:

Using the *trap-obj()* object is equivalent to using the *snmp-obj()* with the **Objectid** type.

version()

Type:	v2c|v3
Default:	v2c

Description: Specifies which version of the SNMP protocol to use.

NOTE:

The syslog-ng PE application will accept any valid option for the *snmp()* destination, but will only use the ones relevant to the selected protocol version, any other option will be ignored. For example, if the **version("v2c") engine-id("0xABABABABAB") community("mycommunity")** options are set, syslog-ng PE will accept every option, but process only the *community()* option, because *engine-id()* applies only to SNMPv3.

Sending messages to a remote log server 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 (for details about the new protocol, see the section called "IETF-syslog messages"). 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 the section called "syslog() destination options".

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, TCP — 601, TLS — 6514.

Example 7.41. Using the syslog() driver

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

Send the log messages using TLS encryption and use mutual authentication. For details on the encryption and authentication options, see the section called "TLS 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')
              )
       );
 };
```

NOTE:

If a message uses the IETF-syslog format (RFC5424), only the text of the message can be customized (that is, the $MESSAGE part of the log), the structure of the header is fixed.

syslog() destination options

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 RFC5424 syslog protocol developed by IETF (for details about the protocol, see the section called "IETF-syslog messages"). The protocol supports sending messages using the RLTP™, UDP, TCP, or the encrypted TLS networking protocols.

These destinations have the following options:

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()	
Type:	yes|no
Default:	no
	Description: If set to **yes**, syslog-ng PE cannot lose logs in case of reload/restart, unreachable destination or

syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to **no**, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of *reliable()* option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored. This option has priority over *--qdisk-dir=*.

CAUTION:

When creating a new *dir()* option for a disk buffer, or modifying an existing one, make sure you delete the persist file, or at least remove the relevant persist-entry.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file or the relevant entry is not deleted after modifying the *dir()* option, then following a restart, syslog-ng PE will look for or create disk-buffer files in their old location. To ensure that syslog-ng PE uses the new *dir()* setting, the persist file must not contain any information about the destinations which the disk-buffer file in question belongs to.

disk-buf-size()

Type:	number (bytes)
Default:	

Description: This is a required option. The maximum size of the disk-buffer in bytes. The minimum value is 1048576 bytes. If you set a smaller value, the minimum value will be used automatically. It replaces the old *log-disk-fifo-size()* option.

mem-buf-length()	
Type:	number (messages)
Default:	10000
	Description: Use this option if the option *reliable()* is set to **no**. This option contains the number of messages stored in overflow queue. It replaces the old *log-fifo-size()* option. It inherits the value of the global *log-fifo-size()* option if provided. If it is not provided, the default value is **10000** messages. Note that this option will be ignored if the option *reliable()* is set to **yes**.

mem-buf-size()	
Type:	number (bytes)
Default:	163840000
	Description: Use this option if the option *reliable()* is set to **yes**. This option contains the size of the messages in bytes that is used in the memory part of the disk buffer. It replaces the old *log-fifo-size()* option. It does not inherit the value of the global *log-fifo-size()* option, even if it is provided. Note that this option will be ignored if the option *reliable()* is set to **no**.

quot-size()	
Type:	number (messages)
Default:	64
	Description: The number of messages stored in the output buffer of the destination.

Options *reliable()* and *disk-buf-size()* are required options.

Example 7.42. Examples for using disk-buffer()

In the following case reliable disk-buffer() is used.

```
destination d_demo {
    network(
            "127.0.0.1"
            port(3333)
            disk-buffer(
```

```
                mem-buf-size(10000)
                disk-buf-size(2000000)
                reliable(yes)
                dir("/tmp/disk-buffer")
            )
        );
};
```

In the following case normal disk-buffer() is used.

```
destination  d_demo  {
       network(
                    "127.0.0.1"
                    port(3333)
                    disk-buffer(
                          mem-buf-length(10000)
                          disk-buf-size(2000000)
                          reliable(no)
                          dir("/tmp/disk-buffer")
                    )
              );
 };
```

failover-servers()

Type:	list of IP addresses and fully-qualified domain names
Default:	0

Description: Available only in syslog-ng Premium Edition version 3.2 and later. Specifies a secondary destination server where log messages are sent if the primary server becomes unaccessible. To list several failover servers, separate the address of the servers with comma. The time syslog-ng PE waits for the a server before switching to the next failover server is set in the *time-reopen()* option. For details about how client-side failover works, see the section called “Client-side failover”.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

NOTE:

This option is not available for the connection-less UDP protocol, because in this case the client does not detect that the destination becomes unaccessible.

Example 7.43. Specifying failover servers for syslog() destinations

The following example specifies two failover servers for a simple syslog() destination.

```
destination d_syslog_tcp{
                syslog("10.100.20.40"
                transport("tcp")
                port(6514)
                failover-servers("10.2.3.4", "myfailoverserver")
                );};
```

The following example specifies a failover server for a network() destination that uses TLS encryption.

```
destination d_syslog_tls{
                network("10.100.20.40"
                transport("tls")
                port(6514)
                failover-servers("10.2.3.4", "myfailoverserver")
                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'))
                );};
```

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The `syslog-protocol` flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. 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, and that the `syslog` driver automatically adds the frame header to the messages.

flush-lines()

Type:	number (messages)
Default:	Use global setting.

Description: Specifies how many lines are sent to a destination at a time. The syslog-ng PE application 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 destination, but also increases message latency.

For optimal performance when sending messages to an syslog-ng PE server, make sure that the *flush-lines()* is smaller than the window size set using the *log-iw-size()* option in the source of your server.

flush-timeout() (OBSOLETE)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush-lines()* option.

NOTE:

This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the *flush-timeout()* option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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 (type of service)
Default:	0

Description: Specifies the Type-of-Service value of outgoing packets.

ip-ttl()

Type:	number (hops)
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 reloaded. 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.

localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to before connecting to target.

localport()

Type:	number (port number)
Default:	0

Description: The port number to bind to. Messages are sent from this port.

log-fifo-size()

Type:	number (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (0), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- **internal**: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:
- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*
- **dst-idle**: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.
- **host-idle**: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **periodical**: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **none**: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.

- **global**: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

port() or localport()

Type:	number (port number)
Default:	UDP — 514, TCP — 601, TLS — 6514.

Description: The port number to connect to.

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. For details, see the **socket(7)** manual page.

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. For details, see the **socket(7)** manual page.

so-rcvbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. For details, see the **socket(7)** manual page.

so-sndbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket send buffer in bytes. For details, see the **socket(7)** manual page.

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. This 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.

CAUTION:

To use spoofing on Microsoft Windows platforms, you must also set the *spoof-interface()* option as well.

When using the *spoof-source* option, syslog-ng PE automatically truncates long messages to 1024 bytes, regardless of the settings of *log-msg-size()*.

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 the section called "Macros of syslog-ng PE". 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.

NOTE:

If a message uses the IETF-syslog format (RFC5424), only the text of the message can be customized (that is, the $MESSAGE part of the log), the structure of the header is fixed.

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backspace characters 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 (messages per second)
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:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various options related to TLS encryption, for example, key/certificate files and trusted CA locations. TLS can be used only with tcp-based transport protocols. For details, see the section called "TLS options".

transport()

Type:	rltp, udp, tcp, or tls
Default:	tcp

Description: Specifies the protocol used to send messages to the destination server.

If you use the **udp** transport, syslog-ng PE automatically sends multicast packets if a multicast destination address is specified. The **tcp** transport does not support multicasting.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global *ts-format()* parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

Sending messages to a remote log server using the legacy BSD-syslog protocol (tcp(), udp() drivers)

NOTE:

The *tcp()*, *tcp6()*, *udp()*, and *udp6()* drivers are obsolete. Use the *network()* source and the *network()* destination instead. For details, see the section called "Collecting messages using the RFC3164 protocol (network() driver)" and the section called "Sending messages to a remote log server using the RFC3164 protocol (network() driver)", respectively.

To convert your existing *tcp()*, *tcp6()*, *udp()*, *udp6()* source drivers to use the *network()* driver, see Procedure 7.8, "Change an old destination driver to the network() driver".

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.

tcp(), tcp6(), udp(), and udp6() destination options

NOTE:

The *tcp()*, *tcp6()*, *udp()*, and *udp6()* drivers are obsolete. Use the *network()* source and the *network()* destination instead. For details, see the section called "Collecting messages using the RFC3164 protocol (network() driver)" and the section called "Sending messages to a remote log server using the RFC3164 protocol (network() driver)", respectively.

To convert your existing *tcp()*, *tcp6()*, *udp()*, *udp6()* source drivers to use the *network()* driver, see Procedure 7.8, "Change an old destination driver to the network() driver".

Procedure 7.8. Change an old destination driver to the network() driver

To replace your existing *tcp()*, *tcp6()*, *udp()*, *udp6()* destinations with a *network()* destination, complete the following steps.

1. Replace the driver with **network**. For example, replace **udp(** with **network(**
2. Set the transport protocol.
   - If you used TLS-encryption, add the **transport("tls")** option, then continue with the next step.
   - If you used the **tcp** or **tcp6** driver, add the **transport("tcp")** option.

- If you used the **udp** or **udp** driver, add the **transport("udp")** option.

3. If you use IPv6 (that is, the **udp6** or **tcp6** driver), add the **ip-protocol("6")** option.
4. If you did not specify the port used in the old driver, check the section called "network() destination options" and verify that your clients send the messages to the default port of the transport protocol you use. Otherwise, set the appropriate port number in your source using the *port()* option.
5. All other options are identical. Test your configuration with the **syslog-ng --syntax-only** command.

   The following configuration shows a simple **tcp** destination.

```
destination  d_old_tcp  {
       tcp(
              "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')
              )
       );
};
```

   When replaced with the **network()** driver, it looks like this.

```
destination  d_new_network_tcp  {
       network(
              "127.0.0.1"
              port(1999)
              transport("tls")
              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')
              )
       );
};
```

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 the section called "unix-stream() and unix-dgram() destination options".

Declaration:

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

Example 7.45. Using the unix-stream() driver

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

unix-stream() and unix-dgram() destination options

These drivers send messages to a unix socket in either *SOCK_STREAM* or *SOCK_DGRAM* mode. The *unix-stream()* and *unix-dgram()* destinations have the following options.

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()	
Type:	yes|no
Default:	no
	Description: If set to **yes**, syslog-ng PE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at

startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of *reliable()* option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored. This option has priority over *--qdisk-dir=*.

CAUTION:

When creating a new *dir()* option for a disk buffer, or modifying an existing one, make sure you delete the persist file, or at least remove the relevant persist-entry.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file or the relevant entry is not deleted after modifying the *dir()* option, then following a restart, syslog-ng PE will look for or create disk-buffer files in their old location. To ensure that syslog-ng PE uses the new *dir()* setting, the persist file must not contain any information about the destinations which the disk-buffer file in question belongs to.

disk-buf-size()

Type:	number (bytes)
Default:	

Description: This is a required option. The maximum size of the disk-buffer in bytes. The minimum value is 1048576 bytes. If you set a smaller value, the minimum value will be used automatically. It replaces the old *log-disk-fifo-size()* option.

mem-buf-length()	
Type:	number (messages)
Default:	10000
	Description: Use this option if the option *reliable()* is set to **no**. This option contains the number of messages stored in overflow queue. It replaces the old *log-fifo-size()* option. It inherits the value of the global *log-fifo-size()* option if provided. If it is not provided, the default value is **10000** messages. Note that this option will be ignored if the option *reliable()* is set to **yes**.

mem-buf-size()	
Type:	number (bytes)
Default:	163840000
	Description: Use this option if the option *reliable()* is set to **yes**. This option contains the size of the messages in bytes that is used in the memory part of the disk buffer. It replaces the old *log-fifo-size()* option. It does not inherit the value of the global *log-fifo-size()* option, even if it is provided. Note that this option will be ignored if the option *reliable()* is set to **no**.

quot-size()	
Type:	number (messages)
Default:	64
	Description: The number of messages stored in the output buffer of the destination.

Options *reliable()* and *disk-buf-size()* are required options.

Example 7.46. Examples for using disk-buffer()

In the following case reliable disk-buffer() is used.

```
destination d_demo {
    network(
            "127.0.0.1"
            port(3333)
            disk-buffer(
```

```
                mem-buf-size(10000)
                disk-buf-size(2000000)
                reliable(yes)
                dir("/tmp/disk-buffer")
            )
        );
};
```

In the following case normal disk-buffer() is used.

```
destination  d_demo  {
       network(
                    "127.0.0.1"
                    port(3333)
                    disk-buffer(
                          mem-buf-length(10000)
                          disk-buf-size(2000000)
                          reliable(no)
                          dir("/tmp/disk-buffer")
                    )
             );
 };
```

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The `syslog-protocol` flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. 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, and that the `syslog` driver automatically adds the frame header to the messages.

flush-lines()

Type:	number (messages)
Default:	Use global setting.

Description: Specifies how many lines are sent to a destination at a time. The syslog-ng PE application 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 destination, but also increases message latency.

For optimal performance when sending messages to an syslog-ng PE server, make sure that the *flush-lines()* is smaller than the window size set using the *log-iw-size()* option in the source of your server.

flush-timeout() (OBSOLETE)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush-lines()* option.

NOTE:

This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the *flush-timeout()* option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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 (messages)
Default:	Use global setting.

Description: The number of messages that the output queue can store.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is reloaded. 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.

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. For details, see the **socket(7)** manual page.

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. For details, see the **socket(7)** manual page.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (0), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- **internal**: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:
- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*
- **dst-idle**: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.
- *MARK* signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.
- **host-idle**: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.
- *MARK* signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.
- **periodical**: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.
- *MARK* signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.
- **none**: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.
- **global**: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

so-rcvbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket receive buffer in bytes. For details, see the **socket(7)** manual page.

so-sndbuf()

Type:	number (bytes)
Default:	0

Description: Specifies the size of the socket send buffer in bytes. For details, see the **socket(7)** manual page.

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 the section called "Macros of syslog-ng PE". 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 for the `'`, `"`, and backspace characters 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 (messages per second)
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:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	Use the global option (which defaults to rfc3164)

Description: Override the global timestamp format (set in the global *ts-format()* parameter) for the specific destination. For details, see the section called "A note on timezones and timestamps".

Sending messages to a user terminal — usertty() destination

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 7.47. Using the usertty() driver

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

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...]);
};
```

CAUTION:

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.

NOTE:

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

Example 8.1. A simple log statement

The following log statement sends all messages arriving to the localhost to a remote server.

```
source s_localhost { network(ip(127.0.0.1) port(1999)); };
destination d_tcp { network("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.

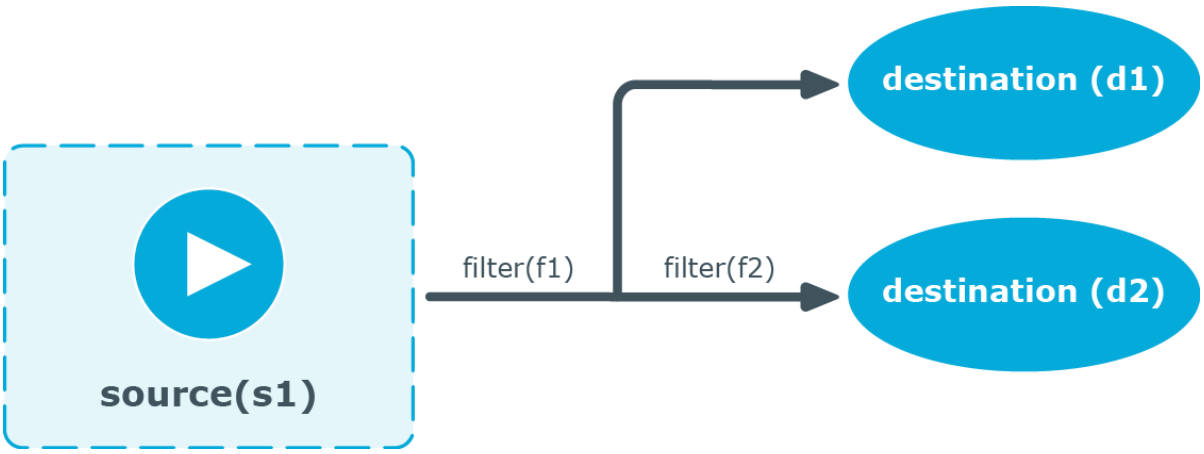
This default behavior can be changed using the `flags()` parameter. Flags apply to individual log paths, they are not global options. For details and examples on the available flags, see the section called “Log path flags”. The effect and use of the `flow-control` flag is detailed in the section called “Managing incoming and outgoing messages with flow-control”.

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.

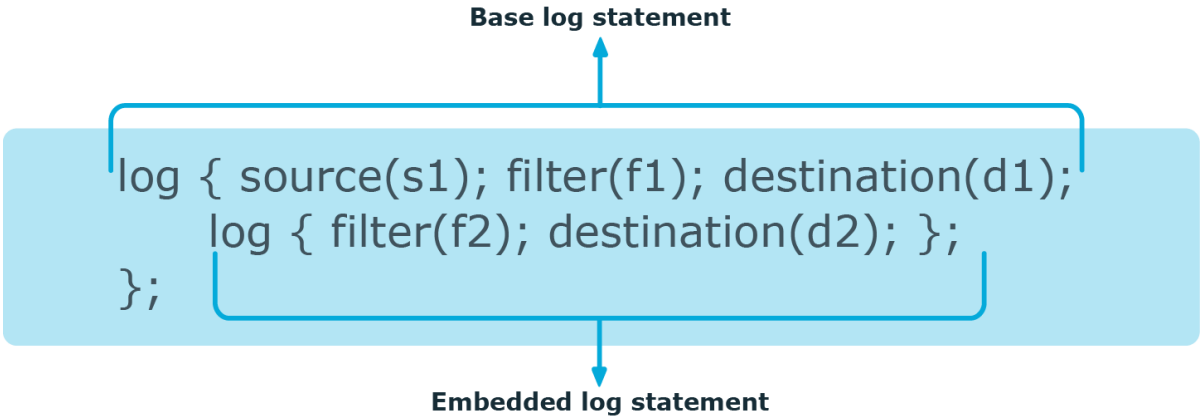
Figure 8.1. Embedded log statement



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 (that is, a top-level log statement can include two or more log statements).
- Embedded log statements can include several levels of log statements (that is, a top-level log statement can include a log statement that includes another log statement, and so on).
- After an embedded log statement, you can write either another log statement, or the `flags()` option of the original log statement. You cannot use filters or other configuration objects. This also means that flags (except for the `flow-control` flag) apply to the entire log statement, you cannot use them only for the embedded log statement.
- 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 8.2. 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.

Using embedded log statements

Embedded log statements (for details, see the section called "Embedded log statements") 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...]);
};
```

Example 8.2. Using embedded log paths

The following log path sends every message to the configured destinations: both the **d_file1** and the **d_file2** destinations receive every message of the source.

```
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 uses two filters:

- messages coming from the host **192.168.1.1** are sent to the **d_file1** destination, and
- messages coming from the host **192.168.1.1** and containing the string **example** are sent to the **d_file2** destination.

```
filter filter_host { host(192.168.1.1) };
filter filter_example { message("example") };
log { source(s_localhost); filter(filter_host); destination(d_file1);
      log {filter(filter_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); };
};
```

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 the section called "Log paths".

Table 8.1. Log statement flags

Flag	Description
catchall	This flag means that the source of the message is ignored, only the filters of the log path 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. 'Processed' means that every filter of a log path matched the message. Note that in case of embedded log paths, the message is considered to be processed if it matches the filters of the outer log path, even if it does not match the filters of the embedded log path. For details, see Example 8.3, "Using log path flags".
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 before the current one (syslog-ng PE evaluates log statements in the order they appear in the configuration file). 'Processed' means that every filter of a log path matched the message. Note that in case of embedded log paths, the message is considered to be processed if it matches the filters of the outer log path, even if it does not match the filters of the embedded log path. For details, see Example 8.3, "Using log path flags".
flow-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. For details, see the section called "Managing incoming and outgoing messages with flow-control".

CAUTION:

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

Example 8.3. 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); };
  ```

The following example shows a scenario that can result in message loss. Do NOT use such a configuration, unless you know exactly what you are doing. The problem is if a message matches the filters in the first part of the first log path, syslog-ng PE treats the message as 'processed'. Since the first log path includes the *final* flag, syslog-ng PE will not pass the message to the second log path (the one with the *fallback* flag). As a result, syslog-ng PE drops messages that do not match the filter of the embedded log path.

```
# Do not use such a configuration, unless you know exactly what you are doing.
log {
    source(s_network);
    # Filters in the external log path.
    # If a message matches this filter, it is treated as 'processed'
    filter(f_program);
    filter(f_message);
    log {
        # Filter in the embedded log path.
        # If a message does not match this filter, it is lost, it will not be
processed by the 'fallback' log path
        filter(f_host);
        destination(d_file1);
```

```
    };
    flags(final);
};

log {
    source(s_network);
    destination(d_file2);
    flags(fallback);
};
```

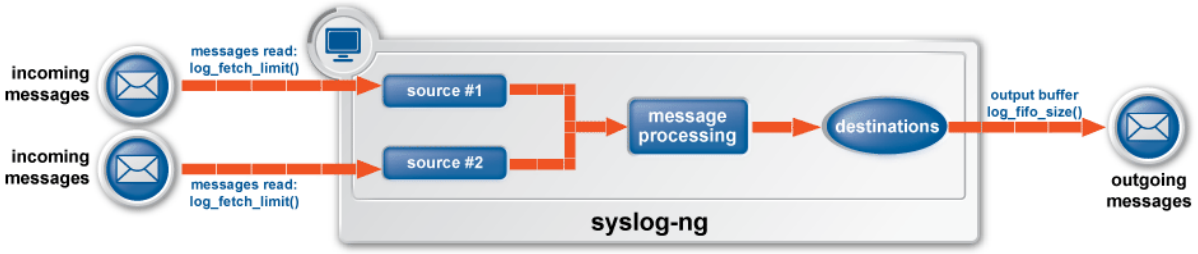
Managing incoming and outgoing messages with flow-control

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 8.3. Managing log messages in syslog-ng



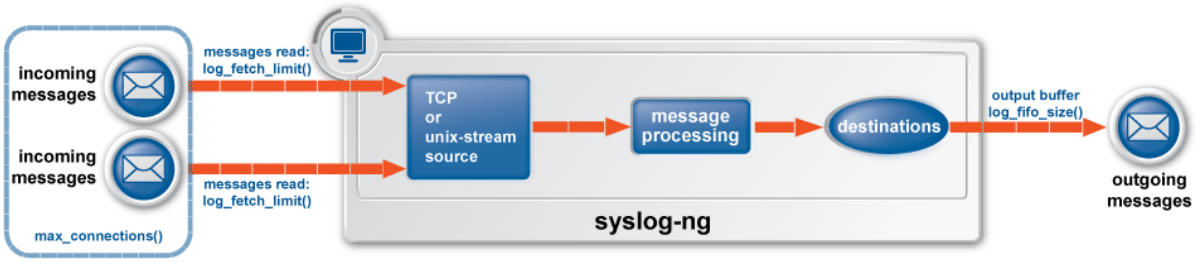
NOTE:

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 8.4. 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 (that is, its size decreases to zero), syslog-ng stops reading messages from the source. The initial size of the control window is by default **1000**: 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.

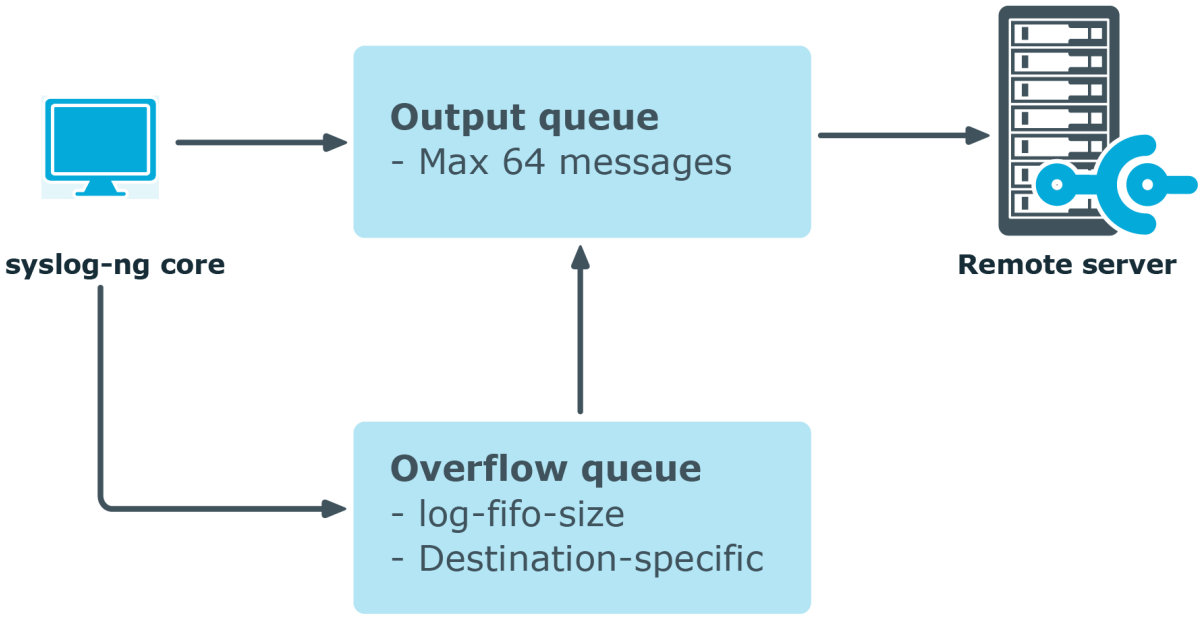
NOTE:

If the source can handle multiple connections (for example, *network()*), the size of the control window is divided by the value of the *max-connections()* parameter and this smaller control window is applied to each connection of the source.

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 8.5. 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. For details on sizing the `log-fifo-size()` parameter, see the section called "Managing incoming and outgoing messages with flow-control".

There are two types of flow-control: Hard flow-control and soft flow-control.

- *Soft flow-control:* In case of soft flow-control there is no message lost if the destination can accept messages, but it is possible to lose messages if it cannot accept messages (for example non-writeable file destination, or the disk becomes full), and all buffers are full. Soft flow-control cannot be configured, it is automatically available for file and logstore destinations.

Example 8.4. Soft flow-control

```
source s_file {file("/tmp/input_file.log");};
destination d_file {file("/tmp/output_file.log");};
destination d_tcp { network("127.0.0.1" port(2222) log-fifo-size(1000));
};
log{ source(s_file); destination(d_file); destination(d_tcp);};
```

CAUTION:

Hazard of data loss! For destinations other than file and logstore, soft flow-control is not available. Thus, it is possible to lose log messages on those destinations. To avoid data loss on those destinations, use hard flow-control.

Hard flow-control: In case of hard flow-control there is no message lost. To use hard flow-control, enable the `flow-control` flag in the log path. Hard flow-control is available for all destinations.

Example 8.5. Hard flow-control

```
source s_file {file("/tmp/input_file.log");};
destination d_file {file("/tmp/output_file.log");};
destination d_tcp { network("127.0.0.1" port(2222) log-fifo-size(1000));
};
log{ source(s_file); destination(d_file); destination(d_tcp) flags(flow-
control);};
```

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.

If you use flow-control and reliable disk-based buffering together with multiple destinations, the flow-control starts slowing down the source only when:

- one destination is down, and
- the number of messages stored in the disk buffer of the destination reaches (*disk-buf-size()* minus *mem-buf-size()*).

Configuring flow-control

For details on how flow-control works, see the section called "Managing incoming and outgoing messages with flow-control". 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, the *log-iw-size()* parameter sets the size of the control window.
- When a source accepts multiple connections, the size of the control window is divided by the value of the *max-connections()* parameter and this smaller control window is applied to each connection of the source.
- 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.

CAUTION:

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 8.6. 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: 1000) 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 {
            network(ip(127.0.0.1) port(1999) max-connections(300)); };
destination d_tcp {
            network("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 {
            network(ip(127.0.0.1) port(1999) max-connections(300)); };
source s_tcp {
            network(ip(192.168.1.5) port(1999) max-connections(100)); };
destination d_tcp {
            network("10.1.2.3" port(1999); localport(999)); log-fifo-size
(40000); };
log { source(s_localhost); destination(d_tcp); flags(flow-control); };
```

Using disk-based and memory buffering

The syslog-ng Premium Edition application can store messages on the local hard disk if the destination (for example, the central log server) or the network connection to the destination becomes unavailable. The syslog-ng PE application automatically sends the stored messages to the destination when the connection is reestablished. The disk buffer is used as a queue: when the connection to the destination is reestablished, syslog-ng PE sends the messages to the destination in the order they were received.

> NOTE:
>
> Disk-based buffering can be used in conjunction with flow-control. For details on flow-control, see the section called "Managing incoming and outgoing messages with flow-control".

The following destination drivers can use disk-based buffering: *elasticsearch()*, *elasticsearch2()*, *hdfs()*, *kafka()*, *mongodb()*, *program()*, *smtp()*, *snmp()*, *sql()*, *unix-dgram()*, and *unix-stream()*. The *network()*, *syslog()*, *tcp()*, and *tcp6()* destination drivers can also use disk-based buffering, except when using the *udp* transport method. (The other destinations or protocols do not provide the necessary feedback mechanisms required for disk-based buffering.)

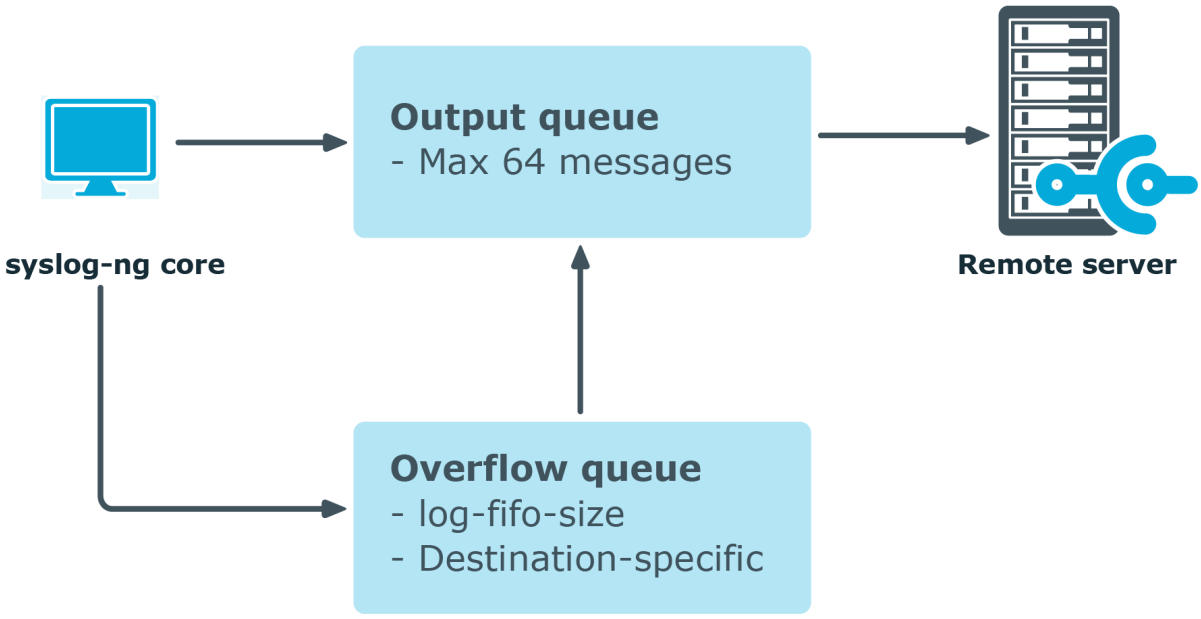
Every such destination uses a separate disk buffer (similarly to the output buffers controlled by *log-fifo-size()*). The hard disk space is not pre-allocated, so ensure that there is always enough free space to store the disk buffers even when the disk buffers are full.

If syslog-ng PE is restarted (using the **/etc/init.d/syslog-ng restart** command, or another appropriate command on your platform), it automatically saves any unsent messages from the disk buffer and the output queue. After the restart, syslog-ng PE sends the saved messages to the destination. In other words, the disk buffer is persistent. The disk buffer is also resistant to syslog-ng PE crashes.

The syslog-ng PE application supports two types of disk buffering: reliable and normal. For details, see the section called "Enabling reliable disk-based buffering" and the section called "Enabling normal disk-based buffering", respectively.

Message handling and normal disk-based buffering. When you use disk-based buffering, and the *reliable()* option is set to **no**, syslog-ng PE handles outgoing messages the following way:

Figure 8.6. Handling outgoing messages in syslog-ng PE



- *Output queue*: Messages from the output queue are sent to the destination (for example, your central log server). The syslog-ng PE application puts the outgoing messages directly into the output queue, unless the output queue is full. By default, the output queue can hold 64 messages (you can adjust it using the *quot-size()* option).
- *Disk buffer*: If the output queue is full, disk-buffering is enabled, and *reliable()* is set to **no**, syslog-ng PE puts the outgoing messages into the disk buffer of the destination. (The disk buffer is enabled if the *log-disk-fifo-size()* parameter of the destination is larger than 0. This option specifies the size of the disk buffer in bytes.)
- *Overflow queue*: If the output queue is full and the disk buffer is disabled or full, syslog-ng PE 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. For details on sizing the *log-fifo-size()* parameter, see also the section called "Managing incoming and outgoing messages with flow-control".

NOTE:

Using disk buffer can significantly decrease performance.

Message handling and reliable disk-based buffering. When you use disk-based buffering, and the *reliable()* option is set to **yes**, syslog-ng PE handles outgoing messages the following way.

The *mem-buf-size()* option determines when flow-control is triggered. All messages arriving to the log path that includes the destination using the disk-buffer are written into the disk-buffer, until the size of the disk-buffer reaches (*disk-buf-size()* minus *mem-buf-size()*). Above that size, messages are written into both the disk-buffer and the

memory-buffer, indicating that flow-control needs to slow down the message source. These messages are not taken out from the control window (governed by *log-iw-size()*), causing the control window to fill up. If the control window is full, the flow-control completely stops reading incoming messages from the source. (As a result, *mem-buf-size()* must be at least as large as *log-iw-size()*.)

Enabling reliable disk-based buffering

The following destination drivers can use disk-based buffering: *elasticsearch()*, *elasticsearch2()*, *hdfs()*, *kafka()*, *mongodb()*, *program()*, *smtp()*, *snmp()*, *sql()*, *unix-dgram()*, and *unix-stream()*. The *network()*, *syslog()*, *tcp()*, and *tcp6()* destination drivers can also use disk-based buffering, except when using the *udp* transport method. (The other destinations or protocols do not provide the necessary feedback mechanisms required for disk-based buffering.)

To enable reliable disk-based buffering, use the **disk-buffer(reliable(yes))** parameter in the destination. Use reliable disk-based buffering if you do not want to lose logs in case of reload/restart, unreachable destination or syslog-ng PE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. The filename of the reliable disk buffer file is the following: `<syslog-ng path>/var/syslog-ng-00000.rqf`.

Example 8.7. Example for using reliable disk-based buffering

```
destination  d_BSD  {
       network(
                     "127.0.0.1"
                     port(3333)
                     disk-buffer(
                           mem-buf-size(10000)
                           disk-buf-size(2000000)
                           reliable(yes)
                     )
              );
};
```

Enabling normal disk-based buffering

The following destination drivers can use disk-based buffering: *elasticsearch()*, *elasticsearch2()*, *hdfs()*, *kafka()*, *mongodb()*, *program()*, *smtp()*, *snmp()*, *sql()*, *unix-dgram()*, and *unix-stream()*. The *network()*, *syslog()*, *tcp()*, and *tcp6()* destination drivers can also use disk-based buffering, except when using the *udp* transport method. (The other destinations or protocols do not provide the necessary feedback mechanisms required for disk-based buffering.)

To enable normal disk-based buffering, use the **`disk-buffer(reliable(no))`** parameter in the destination. Use normal disk-based buffering if you want a solution that is faster than the reliable disk-based buffering. In this case, disk buffering will be less reliable and it is possible to lose logs in case of syslog-ng PE crash. The filename of the normal disk buffer file is the following: `<syslog-ng path>/var/syslog-ng-00000.qf`.

It is possible to use this option without using the disk-buffer plugin. In this case, use the *`log-disk-fifo-size()`* parameter in the destination.

Example 8.8. Example for using normal disk-based buffering

When using the disk-buffer plugin

```
destination d_BSD {
    network(
            "127.0.0.1"
            port(3333)
            disk-buffer(
                mem-buf-length(10000)
                disk-buf-size(2000000)
                reliable(no)
            )
        );
        };
```

Without disk-buffer plugin

```
destination  d_BSD  {
       network(
                    "127.0.0.1"
                    port(3333)
                    log-disk-fifo-size(2000000)
                    log-fifo-size(10000)
             );
 };
```

Enabling memory buffering

To enable memory buffering, use the *`log-fifo-size()`* parameter in the destination. All destination drivers can use memory buffering. Use memory buffering if you want to send logs to destinations where disk-based buffering is not available. Or if you want the fastest solution, and if syslog-ng PE crash or network downtime is never expected. In these cases, losing logs is possible. This solution does not use disk-based buffering, logs are stored only in the memory.

Example 8.9. Example for using memory buffering

```
destination  d_BSD  {
       network(
                    "127.0.0.1"
                    port(3333)
                    log-fifo-size(10000)
             );
 };
```

Client-side failover

syslog-ng PE can detect if the remote server of a network destination becomes unaccessible, and start sending messages to a secondary server. Multiple failover servers can be configured, so if the secondary server becomes unaccessible as well, syslog-ng PE will switch to the third server in the list, and so on. If there are no more failover servers left, syslog-ng PE returns to the beginning of a list and attempts to connect to the primary server.

When syslog-ng PE starts up, it will always try to connect to the primary server first, but once it fails over to a secondary server, it will not automatically attempt to return to the primary server even if it becomes available. If syslog-ng PE is restarted, it will attempt to connect the primary server. Reloading the configuration of syslog-ng PE will not cause syslog-ng PE to return to the primary server, unless the configuration of the destination has changed.

If syslog-ng PE uses TLS-encryption to communicate with the remote server, syslog-ng PE checks the certificate of the failover server as well. The certificates of the failover servers should match their domain names or IP addresses — for details, see the section called "Encrypting log messages with TLS". Note that when mutual authentication is used, the syslog-ng PE client sends the same certificate to every server.

The primary server and the failover servers must be accessible with the same communication method: it is not possible to use different destination drivers or options for the different servers.

NOTE:

Client-side failover works only for TCP-based connections (including TLS-encrypted connections), that is, the *syslog()* and *network()* destination drivers (excluding UDP transport).

Client-side failover is not supported in the *sql()* driver, even though it may use a TCP connection to access a remote database.

For details on configuring failover servers, see the section called "network() destination options" and the section called "syslog() destination options".

Filters

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

- the section called "Using filters" describes how to configure and use filters.
- the section called "Combining filters with boolean operators" shows how to create complex filters using boolean operators.
- the section called "Comparing macro values in filters" explains how to evaluate macros in filters.
- the section called "Using wildcards, special characters, and regular expressions in filters" provides tips on using regular expressions.
- the section called "Tagging messages" explains how to tag messages and how to filter on the tags.
- the section called "Filter functions" is a detailed description of the filter functions available in syslog-ng PE.

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> { <filter_type>("<filter_expression>"); };
```

Example 8.10. 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 {
        source(s1);
        filter(demo_filter);
        destination(d1);};
```

Combining filters with boolean operators

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 {
    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 {
    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")); };
```

For a complete description on filter functions, see the section called “Filter functions”.

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")); };
```

The *value()* parameter of the *match* function limits the scope of the function to the text part of the message (that is, the part returned by the *${MESSAGE}* macro). For details on using the *match()* filter function, see the section called "match()".

TIP:

Filters are often used together with log path flags. For details, see the section called "Log path flags".

Comparing macro values in filters

Starting with syslog-ng PE version 4 F1, it is also possible to compare macro values and templates as numerical and string values. String comparison is alphabetical: it determines if a string is alphabetically greater or equal to another string. Use the following syntax to compare macro values or templates. For details on macros and templates, see the section called "Customizing message format".

```
filter <filter-id>        {"<macro-or-template>" operator "<value-or-macro-or-
template>"};
```

Example 8.11. Comparing macro values in filters

The following expression selects log messages containing a PID (that is, *${PID}* macro is not empty):

```
filter f_pid {"${PID}" !=""};
```

The following expression selects log messages that do not contain a PID. Also, it uses a template as the left argument of the operator and compares the values as strings:

```
filter f_pid {"${HOST}${PID}" eq "${HOST}"};
```

The following example selects messages with priority level 4 or higher.

```
filter f_level {"${LEVEL_NUM}" > "5"};
```

The following filter selects messages which have the collector word in the soc@0.device structured data field.

```
filter f_fwd_collectors {"${.SDATA.soc@0.device}" eq "collector"};
```

This filter expresison selects messages, which has the FW string in the soc@0.class structured data field.

```
filter f_debug { match("FW" value(".SDATA.soc@0.class") type("string")); };
```

Note that:

- The macro or template must be enclosed in double-quotes.
- The **$** character must be used before macros.
- Using comparator operators can be equivalent to using filter functions, but is somewhat slower. For example, using **"${HOST}" eq "myhost"** is equivalent to using **host("myhost" type(string))**.
- You can use any macro in the expression, including user-defined macros from parsers and results of pattern database classifications.
- The results of filter functions are boolean values, so they cannot be compared to other values.
- You can use boolean operators to combine comparison expressions.

The following operators are available:

Table 8.2. Numerical and string comparison operators

Numerical operator	String operator	Meaning
==	eq	Equals
!=	ne	Not equal to
>	gt	Greater than
<	lt	Less than
>=	ge	Greater than or equal
=<	le	Less than or equal

Using wildcards, special characters, and 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. By default, syslog-ng PE uses POSIX regular expressions.

To use other expression types, add the *type()* option after the regular expression. For example:

```
message("^(.+)\\1$" type("pcre"))
```

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`**, and so on. To achieve the wildcard functionality commonly represented by the asterisk character in other applications, use **`.*`** in your expressions, for example **`f.*ilter`**.

Alternatively, if you do not need regular expressions, only wildcards, use **`type(glob)`** in your filter:

Example 8.12. Filtering with widcards

The following filter matches on hostnames starting with the **`myhost`** string, for example, on **`myhost-1`**, **`myhost-2`**, and so on.

```
filter f_wildcard {host("myhost*" type(glob));};
```

For details on using regular expressions in syslog-ng PE, see the section called "Regular expressions".
To filter for special control characters like the carriage return (CR), use the **`\r`** escape prefix in syslog-ng PE version 3.0 and 3.1. In syslog-ng PE 3.2 and later, you can also use the **`\x`** escape prefix and the ASCII code of the character. For example, to filter on carriage returns, use the following filter:

```
filter f_carriage_return {match("\x0d" value ("MESSAGE"));};
```

Tagging messages

You can label the messages with custom tags. Tags are simple labels, identified by their names, which must be unique. Currently syslog-ng PE 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 the section called "Using pattern databases", for details on creating tags in the pattern database, see the section called "The syslog-ng pattern database format".

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 than

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 PE 3.1.1 and later.

For an example on tagging, see Example 8.14, "Adding tags and filtering messages with tags".

Filter functions

The following functions may be used in the filter statement, as described in the section called "Filters".

Table 8.3. Filter functions available in syslog-ng PE

Name	Description
facility()	Filter messages based on the sending facility.
filter()	Call another filter function.
host()	Filter messages based on the sending host.
inlist()	File-based whitelisting and blacklisting.
level() or priority()	Filter messages based on their priority.
match()	Use a regular expression to filter messages based on a specified header or content field.
message()	Use a regular expression to filter messages based on their content.
netmask()	Filter messages based on the IP address of the sending host.
program()	Filter messages based on the sending application.
source()	Select messages of the specified syslog-ng PE source statement.
tags()	Select messages having the specified tag.

facility()

Synopsis: facility(<facility-name>) or facility(<facility-code>) or facility(<facility-name>..<facility-name>)

Description: Match messages having one of the listed facility codes.

The `facility()` filter accepts both the name and the numerical code of the facility or the importance level. Facility codes 0-23 are predefined and can be referenced by their usual name. Facility codes above 24 are not defined.

You can use the facility filter the following ways:

- Use a single facility name, for example, **`facility(user)`**
- Use a single facility code, for example, **`facility(1)`**
- Use a facility range (works only with facility names), for example, **`facility(local0..local5)`**

The syslog-ng application recognizes the following facilities: (Note that some of these facilities are available only on specific platforms.)

Table 8.4. syslog Message Facilities recognized by the facility() filter

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	authpriv	security/authorization messages
11	ftp	FTP daemon
12	ntp	NTP subsystem
13	security	log audit
14	console	log alert
15	solaris-cron	clock daemon
16-23	local0..local7	locally used facilities (local0-local7)

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. Note that you can filter only on the actual content of the HOST field of the

message (or what it was rewritten to). That is, syslog-ng PE will compare the filter expression to the content of the ${HOST} macro. This means that for the IP address of a host will not match, even if the IP address and the hostname field refers to the same host. To filter on IP addresses, use the *netmask()* filter.

```
filter demo_filter { host("example") };
```

inlist()

Synopsis: in-list("</path/to/file.list>", value("<field-to-filter>"));

Description: Matches the value of the specified field to a list stored in a file, allowing you to do simple, file-based black- and whitelisting. The file must be a plain-text file, containing one entry per line. The syslog-ng PE application loads the entire file, and compares the value of the specified field (for example, ${PROGRAM}) to entries in the file. When you use the *in-list filter*, note the following points:

- Comparing the values is case-sensitive.
- Only exact matches are supported, partial and substring matches are not.
- If you modify the list file, reload the configuration of syslog-ng PE for the changes to take effect.

Available in syslog-ng PE and later.

Example 8.13. Selecting messages using the in-list filter

Create a text file that contains the programs (as in the ${PROGRAM} field of their log messages) you want to select. For example, you want to forward only the logs of a few applications from a host: kernel, sshd, and sudo. Create the `/etc/syslog-ng/programlist.list` file with the following contents:

```
kernel
 sshd
 sudo
```

The following filter selects only the messages of the listed applications:

```
filter f_whitelist { in-list("/etc/syslog-ng/programlist.list", value
("PROGRAM")); };
```

Create the appropriate sources and destinations for your environment, then create a log path that uses the previous filter to select only the log messages of the applications you need:

```
log {
source(s_all);
filter(f_whitelist);
destination(d_logserver);};
```

To create a blacklist filter, simply negate the *in-list* filter:

```
filter f_blacklist { not in-list("/etc/syslog-ng/programlist.list", value
("PROGRAM")); };
```

level() or priority()

Synopsis: level(<priority-level>) or level(<priority-level>..<priority-level>)

Description: The *level()* filter selects 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)
```

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 (that is, 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.

The *value()* parameter accepts both built-in macros and user-defined ones created with a parser or using a pattern database. For details on macros and parsers, see the section

called “Templates and macros”, the section called “Parsing messages with comma-separated and similar values”, and the section called “Using parser results in filters and templates”.

message()

Synopsis: message(regexp)

Description: Match a regular expression to the text of the log message, excluding the headers (that is, 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(ipv4/mask)

Description: Select only messages sent by a host whose IP address belongs to the specified IPv4 subnet. Note that this filter checks the IP address of the last-hop relay (the host that actually sent the message to syslog-ng PE), not the contents of the *HOST* field of the message. You can use both the dot-decimal and the CIDR notation to specify the netmask. For example, **192.168.5.0/255.255.255.0** or **192.168.5.0/24**. To filter IPv6 addresses, see the section called “netmask6()”.

netmask6()

Synopsis: netmask6(ipv6/mask)

Description: Select only messages sent by a host whose IP address belongs to the specified IPv6 subnet. Note that this filter checks the IP address of the last-hop relay (the host that actually sent the message to syslog-ng PE), not the contents of the *HOST* field of the message. You can use both the regular and the compressed format to specify the IP address, for example, **1080:0:0:0:8:800:200C:417A** or **1080::8:800:200C:417A**. If you do not specify the address, **localhost** is used. Use the netmask (also called prefix) to specify how many of the leftmost bits of the address comprise the netmask (values 1-128 are valid). For example, the following specify a 60-bit prefix: **12AB:0000:0000:CD30:0000:0000:0000:0000/60** or **12AB::CD30:0:0:0:0/60**. Note that if you set an IP address and a prefix, syslog-ng PE will ignore the bits of the address after the prefix. To filter IPv4 addresses, see the section called “netmask()”.

The *netmask6()* filter is available in syslog-ng PE 5.0.8 and 5.2.2 and later.

CAUTION:

If the IP address is not syntactically correct, the filter will never match. The syslog-ng PE application currently does not send a warning for such configuration errors.

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 8.14. Adding tags and filtering messages with tags

```
source s_tcp {
       network(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");
       };
```

NOTE:

The syslog-ng PE application automatically adds the class of the message as a tag using the *`.classifier.<message-class>`* format. For example, messages classified as "system" receive the *`.classifier.system`* tag. Use the *`tags()`* filter function to

select messages of a specific class.

```
filter f_tag_filter {tags(".classifier.system");};
```

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 8.15. 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); };
```

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 9.1. 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 the section called “Global options”. For important global options and recommendations on their use, see Chapter 20, *Best practices and examples*.

Global options

The following options can be specified in the options statement, as described in the section called “Configuring global syslog-ng options”.

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. If a client sends the log message directly to the syslog-ng PE server, the *chain-hostnames()* option is enabled on the server, and the client sends a hostname in the message that is different from its DNS hostname (as resolved from DNS by the syslog-ng PE server), then the server can append the resolved hostname to the hostname in the message (separated with a **/** character) when the message is written to the destination.

For example, consider a client-server scenario with the following hostnames: *client-hostname-from-the-message*, *client-hostname-resolved-on-the-server*, *server-hostname*. The hostname of the log message written to the destination depends on the *keep-hostname()* and the *chain-hostnames()* options. How *keep-hostname()* and *chain-hostnames()* options are related is described in the following table.

		keep-hostname() setting on the server	
		yes	**no**
chain-hostnames() setting on the server	**yes**	client-hostname-from-the-message	client-hostname-from-the-message / client-hostname-resolved-on-the-server
	no	client-hostname-from-the-message	client-hostname-resolved-on-the-server

If the log message is forwarded to the syslog-ng PE server via a syslog-ng PE relay, the hostname depends on the settings of the *keep-hostname()* and the *chain-hostnames()* options both on the syslog-ng PE relay and the syslog-ng PE server.

For example, consider a client-relay-server scenario with the following hostnames: *client-hostname-from-the-message*, *client-hostname-resolved-on-the-relay*, *client-hostname-resolved-on-the-server*, *relay-hostname-resolved-on-the-server*. How *keep-hostname()* and *chain-hostnames()* options are related is described in the following table.

				chain-hostnames() setting on the server			
				yes		**no**	
				keep-hostname() setting on the server		**keep-hostname() setting on the server**	
				yes	**no**	**yes**	**no**
chain-hostnames() setting on the relay	**yes**	**keep-hostname() setting on the relay**	**yes**	client-hostname-from-the-message	client-hostname-from-the-message / relay-hostname-resolved-on-the- server	client-hostname-from-the-message	relay-hostname-resolved-on-the-server
			no	client-hostname-from-the-message / client-hostname-resolved-on-the-relay	client-hostname-from-the-message / client-hostname-resolved-on-the- relay / relay-hostname-resolved-on-the-server	client-hostname-from-the-message / client-hostname-resolved-on-the-relay	
	no	**keep-hostname() setting on the relay**	**yes**	client-hostname-from-the-message	client-hostname-from-the-message / relay-hostname-resolved-on-the- server	client-hostname-from-the-message	

		chain-hostnames() setting on the server			
		yes		no	
		keep-hostname() setting on the server		keep-hostname() setting on the server	
		yes	no	yes	no
	no	client-hostname-resolved-on-the-relay	client-hostname-resolved-on-the- relay / relay-hostname-resolved-on-the-server	client-hostname-resolved-on-the-relay	

The *chain-hostnames()* option of syslog-ng can interfere with the way syslog-ng PE counts the log source hosts, causing syslog-ng to think there are more hosts logging to the central server, especially if the clients sends a hostname in the message that is different from its real hostname (as resolved from DNS). Disable the *chain-hostnames()* option on your log sources to avoid any problems related to license counting.

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.

custom-domain()

Accepted values:	string
Default:	**empty string**

Description: Use this option to specify a custom domain name that is appended after the short hostname to receive the FQDN. This option affects every outgoing message: eventlog sources, file sources, MARK messages and internal messages of syslog-ng PE.

- If the hostname is a short hostname, the custom domain name is appended after the hostname (for example **mypc** becomes **mypc.customcompany.local**).
- If the hostname is an FQDN, the domain name part is replaced with the custom domain name (for example if the FQDN in the forwarded message is **mypc.mycompany.local** and the custom domain name is **customcompany.local**, the hostname in the outgoing message becomes **mypc.customcompany.local**).

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 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 also the *create-dirs()* option). For octal numbers prefix the number with **0**, for example use **0755** for **rwxr-xr-x**.

To preserve the original properties of an existing directory, use the option without specifying an attribute: *dir-perm()*. Note that when creating a new directory without specifying attributes for *dir-perm()*, the default permission of the directories is masked with the umask of the parent process (typically **0022**).

dns-cache()

Accepted values:	**yes** | **no**
Default:	**yes**

Description: Enable or disable DNS cache usage.

NOTE:

This option has no effect if the *keep-hostname()* option is enabled (**keep-hostname (yes)**) and the message contains a hostname.

dns-cache-expire()

Accepted values:	number (seconds)
Default:	3600

Description: Number of seconds while a successful lookup is cached.

dns-cache-expire-failed()

Accepted values:	number (seconds)
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 of hostnames
Default:	1007

Description: Number of hostnames in the DNS cache.

file-template()

Accepted values:	string
Default:	

Description: Specifies a template that file-like destinations use by default. For example:

```
template t_isostamp { template("$ISODATE $HOST $MSGHDR$MSG\n"); };

options { file-template(t_isostamp); };
```

flush-lines()

Accepted values:	number (messages)
Default:	1

Description: Specifies how many lines are flushed to a destination at a time. The syslog-ng PE application 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.

flush-timeout() (OBSOLETE)

Accepted values:	time in milliseconds
Default:	10000

Description: This is an obsolete option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush-lines()* option.

> NOTE:
>
> This option will be removed from the list of acceptable options. After that, your configuration will become invalid if it still contains the *flush-timeout()* option. To avoid future problems, remove this option from your configuration.

frac-digits()

Type:	number (digits of fractions of a second)
Default:	Value of the global option (which defaults to 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()

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.

- If enabled (**`keep-hostname(yes)`**), syslog-ng PE assumes that the incoming log message was sent by the host specified in the *`HOST`* field of the message.
- If disabled (**`keep-hostname(no)`**), syslog-ng PE rewrites the *`HOST`* field of the message, either to the IP address (if the *`use-dns()`* parameter is set to **`no`**), or to the hostname (if the *`use-dns()`* parameter is set to **`yes`** and the IP address can be resolved to a hostname) of the host sending the message to syslog-ng PE. For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

NOTE:

If the log message does not contain a hostname in its *`HOST`* field, syslog-ng PE automatically adds a hostname to the message.

- For messages received from the network, this hostname is the address of the host that sent the message (this means the address of the last hop if the message was transferred via a relay).
- For messages received from the local host, syslog-ng PE adds the name of the host.

This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

When relaying messages, enable this option on the syslog-ng PE server and also on every relay, otherwise syslog-ng PE 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.

CAUTION:

To use the `S_` macros, the *`keep-timestamp()`* option must be enabled (this is the default behavior of syslog-ng PE).

log-fifo-size()

Accepted values:	number (messages)
Default:	10000

Description: The number of messages that the output queue can store.

log-msg-size()

Accepted values:	number (bytes)
Default:	65535

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

logstore-journal-shmem-threshold()

Type:	number (bytes)
Default:	536870912

Description: If the size of memory (in bytes) required by journal files increases above this value, syslog-ng PE maps only a single block of every logstore journal into the memory. Default value: **536870912** (512 MB).

If the memory required for the journal files exceeds the *`logstore-journal-shmem-threshold()`* limit, syslog-ng PE will store only a single journal block of every journal file in the memory, and — if more blocks are needed for a journal — store the additional blocks on the hard disk. Opening new logstore files means allocating memory for one new journal

block for every new file. In extreme situations involving large traffic, this can lead to syslog-ng PE consuming the entire memory of the system. Adjust the *journal-block-size()* and your file-naming conventions as needed to avoid such situations. For details on logstore journals, see the section called "Journal files".

Example 9.2. Calculating memory usage of logstore journals

If you are using the default settings (4 journal blocks for every logstore journal, one block is 1MB, *logstore-journal-shmem-threshold()* is 512MB), this means that syslog-ng PE will allocate 4MB memory for every open logstore file, up to 512MB if you have 128 open logstore files. Opening a new logstore file would require 4 more megabytes of memory for journaling, bringing the total required memory to 516MB, which is above the *logstore-journal-shmem-threshold()*. In this case, syslog-ng PE switches to storing only a single journal block in the memory, lowering the memory requirements of journaling to 129MB. However, opening more and more logstore files will require more and more memory, and this is not limited, except when syslog-ng PE reaches the maximum number of files that can be open (as set in the *--fd-limit* command-line option).

Example 9.3. Limiting the memory use of journal files

The following example causes syslog-ng PE to map only a single journal block into the host's memory if the total memory range used by logstore journals would be higher than 32 MB.

```
options  {
       logstore-journal-shmem-threshold(33554432);
 };
 destination  d_messages  {  logstore("/var/log/messages_logstore.lgs"
                                      journal-block-size(19660800)
                                      journal-block-count(5)
                                      );
 };
```

mark() (DEPRECATED)

Accepted values:	number (seconds)
Default:	1200

Description: The *mark-freq()* option is an alias for the deprecated *mark()* option. This is retained for compatibility with syslog-ng version 1.6.x.

mark-freq()

Accepted values:	number (seconds)
Default:	1200

Description: An alias for the obsolete *mark()* option, retained for compatibility with syslog-ng version 1.6.x. 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. If set to zero (**0**), no *MARK* messages are sent. The *mark-freq()* can be set for global option and/or every *MARK* capable destination driver if *mark-mode()* is periodical or dst-idle or host-idle. If *mark-freq()* is not defined in the destination, then the *mark-freq()* will be inherited from the global options. If the destination uses internal *mark-mode()*, then the global *mark-freq()* will be valid (does not matter what *mark-freq()* set in the destination side).

mark-mode()

Accepted values:	**internal** | **dst-idle** | **host-idle** | **periodical** | **none** | **global**
Default:	**internal** for pipe, program drivers **none** for file, unix-dgram, unix-stream drivers **global** for syslog, tcp, udp destinations **host-idle** for global option

Description: The *mark-mode()* option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- **internal**: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng PE 3.x worked. *MARK* will be generated by internal source if there was NO traffic on local sources:
- *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*, *program()*
- **dst-idle**: Sends *MARK* signal if there was NO traffic on destination drivers. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **host-idle**: Sends *MARK* signal if there was NO local message on destination drivers. For example *MARK* is generated even if messages were received from tcp. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **periodical**: Sends *MARK* signal perodically, regardless of traffic on destination driver. *MARK* signal from internal source will be dropped.

  MARK signal can be sent by the following destination drivers: *network()*, *syslog()*, *program()*, *file()*, *pipe()*, *unix-stream()*, *unix-dgram()*.

- **none**: Destination driver drops all *MARK* messages. If an explicit mark-mode() is not given to the drivers where **none** is the default value, then **none** will be used.

- **global**: Destination driver uses the global *mark-mode()* setting. The syslog-ng interprets syntax error if the global *mark-mode()* is global.

NOTE:

In case of **dst-idle**, **host-idle** and **periodical**, the *MARK* message will not be written in the destination, if it is not open yet.

Available in syslog-ng PE 4 LTS and later.

normalize-hostnames()

Accepted values:	**yes** | **no**
Default:	**no**

Description: If enabled (**normalize-hostnames(yes)**), syslog-ng PE converts the hostnames to lowercase.

NOTE:

This setting applies only to hostnames resolved from DNS. It has no effect if the *keep-hostname()* option is enabled, and the message contains a hostname.

on-error()

Accepted values:	drop-message|drop-property|fallback-to-string|silently-drop-message|silently-drop-property|silently-fallback-to-string
Default:	drop-message

Description: Controls what happens when type-casting fails and syslog-ng PE cannot convert some data to the specified type. By default, syslog-ng PE drops the entire message and logs the error. Currently the *value-pairs()* option uses the settings of *on-error()*.

- *drop-message*: Drop the entire message and log an error message to the *internal()* source. This is the default behavior of syslog-ng PE.
- *drop-property*: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the *internal()* source.
- *fallback-to-string*: Convert the property to string and log an error message to the *internal()* source.
- *silently-drop-message*: Drop the entire message silently, without logging the error.
- *silently-drop-property*: Omit the affected property (macro, template, or message-field) silently, without logging the error.
- *silently-fallback-to-string*: Convert the property to string silently, without logging the error.

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.

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.

proto-template()

Accepted values:	name of a template
Default:	The default message format of the used protocol

Description: Specifies a template that protocol-like destinations (for example, network() and syslog()) use by default. For example:

```
template t_isostamp { template("$ISODATE $HOST $MSGHDR$MSG\n"); };

options { proto-template(t_isostamp); };
```

recv-time-zone()

Accepted values:	name of the timezone, or the timezone offset
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. For details, see also the section called "Timezones and daylight saving" and the section called "A note on timezones and timestamps".

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

send-time-zone()

Accepted values:	name of the timezone, or the timezone offset
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. For details, see the section called "Timezones and daylight saving".

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

The timezone can be specified as using the name of the (for example **`time-zone ("Europe/Budapest")`**), or as the timezone offset in +/-HH:MM format (for example **`+01:00`**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

stats-freq()

Accepted values:	number (seconds)
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. For details on message statistics, see Chapter 17, *Statistics and metrics of syslog-ng*.

sync() or sync-freq() (DEPRECATED)

Accepted values:	number (messages)
Default:	0

Description: Obsolete aliases for *`flush-lines()`*

threaded()

Accepted values:	yes|no
Default:	yes

Description: Enable syslog-ng PE to run in multithreaded mode and use multiple CPUs. Available only in syslog-ng Premium Edition 4 F1 and later. See Chapter 18, *Multithreading and scaling in syslog-ng PE* for details.

time-reap()

Accepted values:	number (seconds)
Default:	60

Description: The time to wait in seconds before an idle destination file is closed. Note that only destination files having macros in their filenames are closed automatically.

time-reopen()

Accepted values:	number (seconds)
Default:	60

Description: The time to wait in seconds before a dead connection is reestablished.

time-sleep() (DEPRECATED)

Accepted values:	number (milliseconds)
Default:	0

Description: The time to wait in milliseconds between each invocation of the *poll()* iteration.

timestamp-freq()

Type:	number (seconds)
Default:	0

Description: The minimum time (in seconds) that should expire between two timestamping requests. When syslog-ng closes a chunk, it checks how much time has expired since the last timestamping request: if it is higher than the value set in the *timestamp-freq()* parameter, it requests a new timestamp from the authority set in the *timestamp-url()* parameter.

By default, timestamping is disabled: the *timestamp-freq()* global option is set to 0. To enable timestamping, set it to a positive value.

timestamp-url()

Accepted values:	string
Default:	

Description: The URL of the Timestamping Authority used to request timestamps to sign logstore chunks. Note that syslog-ng PE currently supports only Timestamping Authorities that conform to *RFC3161 Internet X.509 Public Key Infrastructure Time-Stamp Protocol*, other protocols like *Microsoft Authenticode Timestamping* are not supported.

timestamp-policy()

Accepted values:	string
Default:	

Description: If the Timestamping Server has timestamping policies configured, specify the OID of the policy to use into the Timestamping policy field. syslog-ng PE will include this ID in the timestamping requests sent to the TSA. This option is available in syslog-ng PE 3.1 and later.

time-zone()

Type:	name of the timezone, or the timezone offset
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. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, *HOUR*. For the complete list of such macros, see the section called "Date-related macros".

The timezone can be specified as using the name of the (for example **time-zone ("Europe/Budapest")**), or as the timezone offset in +/-HH:MM format (for example **+01:00**). On Linux and UNIX platforms, the valid timezone names are listed under the `/usr/share/zoneinfo` directory.

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). For details, see also the section called "A note on timezones and timestamps".

By default, timestamps include only seconds. To include fractions of a second (for example, milliseconds) use the *frac-digits()* option. For details, see the section called "frac-digits()".

NOTE:

This option applies only to file and file-like destinations. Destinations that use specific protocols (for example, *network()*, or *syslog()*) ignore this option. For protocol-like destinations, use a template locally in the destination, or use the *proto-template* option.

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`). The syslog-ng PE application 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.

NOTE:

This option has no effect if the *keep-hostname()* option is enabled (**keep-hostname (yes)**) and the message contains a hostname.

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.

NOTE:

This option has no effect if the *keep-hostname()* option is enabled (**keep-hostname (yes)**) and the message contains a hostname.

use-rcptid() (DEPRECATED)

Accepted values:	**yes** | **no**
Default:	**no**

Description: The receipt ID function is disabled by default due to performance issues. For details, see also the section called "RCPTID". This function is now deprecated. Use the *use-uniqid()* option instead, for details, see the section called "use-uniqid()".

use-time-recvd() (DEPRECATED)

Accepted values:	**yes** | **no**
Default:	**no**

CAUTION:

This option is not available in syslog-ng PE version 3.2 and later. Use the R_ prefixed version of the respective macro instead. Starting with syslog-ng PE version 3.2, the *DATE* macro equals the *S_DATE* macro.

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.

use-uniqid()

Accepted values:	**yes** | **no**
Default:	**no**

Description: This option enables generating a globally unique ID. It is generated from the HOSTID and the RCPTID in the format of HOSTID@RCPTID. It has a fixed length: 16+@+8 characters. You can include the unique ID in the message by using the macro. For details, see the section called "UNIQID".

Enabling this option automatically generates the HOSTID. The HOSTID is a persistent, 32-bits-long cryptographically secure pseudo random number, that belongs to the host that the syslog-ng is running on. If the persist file is damaged, the HOSTID might change.

Enabling this option automatically enables the RCPTID functionality. For details, see the section called "RCPTID"

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 using the *network()* and *syslog()* drivers.

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

Figure 10.1. 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 (or the self-signed 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.

For details on configuring TLS communication in syslog-ng, see the section called "Encrypting log messages with TLS".

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 the section called "Secure logging using TLS".

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.

Procedure 10.1. Configuring TLS on the syslog-ng clients

Purpose:

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

Steps:

1. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng server (or the self-signed 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.

   Note that by default, syslog-ng PE expects SHA-1 hashes. If you want to use MD5 hashes for some reason, use the **ca_dir_layout(md5-based)** option in your configuration (for details, see *Collecting log messages from UDP sources*).

   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.

2. **ln -s cacert.pem 6d2962a8.0**

   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 *network()* or the *syslog()* destination driver, and the IP address and port parameters of the driver must point to the syslog-ng server.

Example 10.1. 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 {
       network("10.1.2.3" port(6514)
              transport("tls")
              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")
              tls(ca-dir("/opt/syslog-ng/etc/syslog-ng/ca.d"))
       );
};
```

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

CAUTION:

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.

Procedure 10.2. Configuring TLS on the syslog-ng server

Purpose:

Complete the following steps on the syslog-ng server:

Steps:

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.
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.
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 (*`network()`* or *`syslog()`*) matching the destination driver used by the syslog-ng client.

Example 10.2. 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  {
       network(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"))
       );
 };
```

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"))
       );
 };
```

4.

Disable mutual authentication for the source by setting the following TLS option in the source statement: **`tls( peer-verify(optional-untrusted);`**

For details on how to configure mutual authentication, see the section called “Mutual authentication using TLS”.

For the details of the available *`tls()`* options, see the section called “TLS options”.

Example 10.3. 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  {
       network(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))
       );
 };
```

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))
       );
 };
```

CAUTION:

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

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 (for details, see the section called "Encrypting log messages with TLS"), 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 the section called "Secure logging using TLS".

Procedure 10.3. Configuring TLS on the syslog-ng clients

Purpose:

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

Steps:

1. Create an X.509 certificate for the syslog-ng client.
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.
3. Copy the CA certificate of the Certificate Authority (for example `cacert.pem`) that issued the certificate of the syslog-ng server (or the self-signed 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.

   Note that by default, syslog-ng PE expects SHA-1 hashes. If you want to use MD5 hashes for some reason, use the **ca_dir_layout(md5-based)** option in your configuration.

   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.

4. **ln -s cacert.pem 6d2962a8.0**

   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 *network()* or the *syslog()* 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 10.4. 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 {
    network("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")) ); };
```

```
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")) ); };
```

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

CAUTION:

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.

Procedure 10.4. Configuring TLS on the syslog-ng server

Purpose:

Complete the following steps on the syslog-ng server:

Steps:

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.
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.

Note that by default, syslog-ng PE expects SHA-1 hashes. If you want to use MD5 hashes for some reason, use the **`ca_dir_layout(md5-based)`** option in your configuration.

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

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.
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 (*`network()`* or *`syslog()`*) 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.

   For the details of the available *`tls()`* options, see the section called “TLS options”.

Example 10.5. 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 {
    network(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")) ); };
```

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")) ); };
```

CAUTION:

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

TLS options

The syslog-ng application can encrypt incoming and outgoing syslog message flows using TLS if you use the *network()* or *syslog()* drivers.

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 *transport("tls")* and *tls()* options in the source and destination statements.

The tls() option can include the following settings:

allow-compress()

Accepted values:	yes | no
Default:	no

Description: Enable on-the-wire compression in TLS communication. Note that this option must be enabled both on the server and the client side to have any effect. Enabling compression can significantly reduce the bandwidth required to transport the messages, but can slightly decrease the performance of syslog-ng PE, reducing the number of transferred messages during a given period.

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 have 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.

ca-dir-layout() (DEPRECATED)

Accepted values:	**sha1-based**
Default:	**sha1-based**

Description: The type of the hash used for the CA certificates. NOTE: This option is deprecated.

CAUTION:

If you are upgrading to syslog-ng PE version 6.x from a version earlier than 5.0, you must rehash the trusted CA certificates.

cert-file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate (or a certificate chain) in PEM format, suitable as a TLS certificate, matching the private key. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	1.1.1

Description: Specifies the cipher, hash, and key-exchange algorithms used for the encryption, for example, `ECDHE-ECDSA-AES256-SHA384`. The list of available algorithms depends on the version of OpenSSL used to compile syslog-ng PE. To specify multiple ciphers, separate the cipher names with a colon, and enclose the list between double-quotes, for example:

```
cipher-suite("ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384")
```

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`**.

If the *`crl-dir()`* is set, and the peer certificate has been revoked, syslog-ng PE rejects the connection. If the peer certificate has not been revoked, or syslog-ng PE cannot access the CRL, syslog-ng PE accepts the connection.

curve-list()

Accepted values:	string (colon-separated list)
Default:	none

Description: A colon-separated list that specifies the curves that are permitted in the connection when using Elliptic Curve Cryptography (ECC). The syslog-ng PE application uses automatically the highest preference curve that both peers support. If not specified, the list includes every supported curve. For example:

```
curve-list('prime256v1:secp521r1')
```

The syslog-ng Premium Edition application currently supports the following curves: **sect163k1**, **sect163r1**, **sect163r2**, **sect193r1**, **sect193r2**,, **sect233k1**, **sect233r1**, **sect239k1**, **sect283k1**, **sect283r1**,, **sect409k1**, **sect409r1**, **sect571k1**, **sect571r1**, **secp160k1**,, **secp160r1**, **secp160r2**, **secp192k1**, **prime192v1**, **secp224k1**,, **secp224r1**, **secp256k1**, **prime256v1**, **secp384r1**, **secp521r1**,, **brainpoolP256r1**, **brainpoolP384r1**, **brainpoolP512r1**.

dhparam-file()

Accepted values:	string (filename)
Default:	none

Description: Specifies a file containing Diffie-Hellman parameters, generated using the **openssl dhparam** utility. Note that syslog-ng PE supports only DH parameter files in the PEM format. If you do not set this parameter, syslog-ng PE uses the 2048-bit MODP Group, as described in RFC 3526.

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 valid or not.

The following table summarizes the possible options and their results depending on the certificate of the peer.

		The remote peer has:		
		no certificate	**invalid certificate**	**valid certificate**
Local peer-verify() setting	**optional-untrusted**	TLS-encryption	TLS-encryption	TLS-encryption
	optional-trusted	TLS-encryption	rejected connection	TLS-encryption
	required-untrusted	rejected connection	TLS-encryption	TLS-encryption
	required-trusted	rejected connection	rejected connection	TLS-encryption

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 mismatches.

CAUTION:

When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, syslog-ng PE will reject the connection.

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")`.

To find the fingerprint of a certificate, you can use the following command: **openssl x509 -in <certificate-filename> -sha1 -noout -fingerprint**

NOTE:

When using the `trusted-keys()` and `trusted-dn()` parameters at the same time, note the following:

- If the fingerprint of the peer is listed in the `trusted-keys()` parameter and the DN of the peer is listed in the `trusted-dn()` parameter, then the certificate validation is performed.
- If either the fingerprint of the peer is not listed in the `trusted-keys()` parameter or the DN of the peer is not listed in the `trusted-dn()` parameter, then the authentication of the peer fails and the connection is closed.

Logging using RLTP™

The syslog-ng PE application can send and receive log messages in a reliable way over the TCP transport layer using the Reliable Log Transfer Protocol™ (RLTP™). RLTP™ is a proprietary transport protocol that prevents message loss during connection breaks. The transport is used between syslog-ng PE hosts (for example, a client and a server, or a client-relay-server), and interoperates with the flow-control and reliable disk-buffer mechanisms of syslog-ng PE, thus providing the best way to prevent message loss. The sender detects which messages has the receiver successfully received. If messages are lost during the transfer, the sender resends the missing messages, starting from the last successfully received message. Therefore, messages are not duplicated at the receiving end in case of a connection break (however, in failover mode this is not completely ensured). RLTP™ also allows to receive encrypted and non-encrypted connections on the same port, using a single source driver.

NOTE:

Because of the communication overhead, the RLTP™ protocol is slower than other transport protocols, which might be a problem if you need to collect a high amount (over 200000 messages per second) of log messages on your log server. For performance details of syslog-ng PE see the *syslog-ng Premium Edition Performance Guideline* at the syslog-ng Documentation page.

NOTE:

Make sure that you have set the value of the `log_msg_size()` parameter large enough in your configuration. If its size is less than the size of the sent messages, it might result in disk fill-up and no incoming logs.

CAUTION:

In the following cases, it is possible to lose log messages even if you use RLTP™:

- **If you use RLTP™ together with non-reliable disk-buffer, it is possible to lose logs.**
- **When sending logs through a relay that is using a non-reliable disk-buffer, it is possible to lose logs if the relay crashes.**
- **When sending logs through a relay that is using a non-reliable disk-buffer, it is possible that logs are duplicated if the relay crashes, or it is stopped.**
- **If the underlying disk system of syslog-ng PE fails to write the log messages to the disk, but it does not return a write error, or some other hardware or operating-system error happens.**

The RLTP™ protocol works on top of TCP, and can use STARTTLS for encryption. RLTP™ supports IPv4 and IPv6 addresses. Inside the RLTP™ message, the message can use any format, for example, RFC3164 (BSD-syslog) or RFC5424 (IETF-syslog). The default port of RLTP™ is **35514**.

RLTP™ can be added to the configuration like a transport protocol within the *syslog()* driver and the *network()* driver.

Procedure 12.1. How RLTP™ connections work

Purpose:

This procedure summarizes how two syslog-ng PE hosts (a sender and a receiver) communicate using the Reliable Log Transfer Protocol™ (RLTP™).

Prerequisites:

The sender (also called the client) is the host that has RLTP™ configured in its destination driver. The receiver (also called the server) is the host that has RLTP™ configured in its source driver.

Steps:

1. The sender initiates the connection to the receiver.
2. The sender and the receiver negotiate whether to encrypt the connection and to use compression or not.
3. If the connection should be encrypted, the sender and the receiver perform authentication (as configured in the *tls()* options of their configuration).
4. If the sender and the receiver have communicated earlier using RLTP™, the receiver indicates which was the last message received from the sender.
5. The sender starts sending messages in batches. Batch size depends on the *flush-lines()* parameter of the sender.

   For optimal performance when sending messages to an syslog-ng PE server, make sure that the *flush-lines()* is smaller than the window size set using the *log-iw-size()* option in the source of your server.
6. When the receiver has successfully processed the messages in the batch, it sends an acknowledgement of the processed messages to the sender.

   What "successfully processed" means depends on the configuration of the receiver, for example, written to disk in a destination, forwarded to a remote destination using *not*RLTP™, dropped because of filter settings, or written to the disk-buffer. (If the messages are forwarded using RLTP™, see the section called “Using RLTP™ in a client-relay-server scenario”.)
7. After receiving the acknowledgement, the sender sends another batch of messages.

Using RLTP™ in a client-relay-server scenario

You can use RLTP™ between multiple syslog-ng PE hosts, for example, in a client-relay-server scenario. In such case, the communication described in Procedure 12.1, “How

RLTP™ connections work" applies both between the client and the relay, and the relay and the server. However, note the following points:

- Unless you use disk-buffer on the relay, the relay waits for acknowledgement from the server before acknowledging the messages to the client. If you send the messages in large batches, and the server can process the messages slowly (or the network connection is slow), you might have to adjust the *message-acknowledgement-timeout()* on the client.
- If you use reliable disk-buffer on the relay, the relay will acknowledge the messages when the messages are written to the disk-buffer. That way, the client does not have to wait while the server acknowledges the messages.

RLTP™ options

The following options are specific to the RLTP™ protocol. Note that when using RLTP™ in a source or a destination, the options of the *syslog()* or the *network()* driver can be used as well.

allow-compress()

Accepted values:	yes | no
Default:	no

Description: Enable on-the-wire compression in the RLTP communication. Note that this option must be enabled both on the server and the client side to have any effect. Enabling compression can significantly reduce the bandwidth required to transport the messages, but can slightly decrease the performance of syslog-ng PE, reducing the number of transferred messages. The *allow-compress()* option can be used in source and destination drivers as well. Available in syslog-ng PE 5.0 and later.

message-acknowledgement-timeout()

Type:	number (seconds)
Default:	900

Description: When the receiver (syslog-ng PE server) receives and successfully processes a message, it sends an acknowledgement to the sender (the syslog-ng PE client). If the receiver does not acknowledge receiving the messages within this period, the sender terminates the connection with the receiver. Use this option only in destination drivers.

response-timeout()

Type:	number (seconds)
Default:	60

Description: If syslog-ng PE does not receive any protocol-related message in the given timeframe (except for message acknowledgement, which is governed by the `message-acknowledgement-timeout()` option), syslog-ng PE terminates the connection with the peer, and the "Connection broken" message appears in the logs of the sender (the syslog-ng PE client). This is normal, and happens when the sender does not send any new message to the receiver.

Under normal circumstances, you should not change the value of this option. The *response-timeout()* option can be used in source and destination drivers as well.

tls-required()

Type:	yes, optional, no
Default:	optional

Description: Determines whether STARTTLS is to be used during communication. If the option is set to **yes**, you must also configure the *tls()* option to specify other parameters of the TLS connection (for example, the authentication of the server and the client).

The *tls-required()* option can be used in source and destination drivers as well.

For example, if you configure **tls-required(yes)** on server side and **tls-required(no)** on client side, the connection is dropped. If one of them is set to **optional**, the configuration of the other side will decide if TLS is used or not. If both sides are set to **optional**, and the *tls()* option is properly configured, TLS encryption will be used. The following table summarizes the possible options and their results.

Note that the various parameters of the *tls()* option are considered in the connection only if the *tls-required()* settings of the peers result in **TLS-encryption** in the following table. In other words: the *tls-required()* option of RLTP™ determines if TLS should be used at all, while the *peer-verify()* option of the *tls()* setting determines if the TLS connection can be actually established.

		tls-required() setting on the server		
		yes	**no**	**optional**
tls-required() setting on the client	**yes**	TLS-encryption	rejected connection	TLS-encryption
	no	rejected connection	unencrypted connection	unencrypted connection
	optional	TLS-encryption	unencrypted connection	TLS-encryption if the *tls()* option is set, unencrypted connection otherwise

Setting **tls-required(optional)** on your server allows you to receive both encrypted and unencrypted connections on the same port.

Examples for using RLTP™

Example 12.1. Simple RLTP™ connection

The sender and the receiver use RLTP™ over the *network()* protocol. Since the **tls()** option is not configured neither on the sender nor on the receiver, the communication will be unencrypted.

Receiver configuration (syslog-ng PE server):

```
source  s_network_rltp  {
              network(
                    ip("127.0.0.1")
                    port("5555")
                    transport(rltp)
                    ip-protocol(4)
              );
 };
```

Sender configuration (syslog-ng PE client):

```
destination  d_network_rltp  {
              network(
                    "127.0.0.1"
                    port("5555")
                    transport(rltp)
                    ip-protocol(4)
              );
 };
```

Example 12.2. RLTP™ with TLS encryption

The following example configure a sender and a receiver to communicate using RLTP™. Since the *tls-required()* option is set to **optional** on the receiver and **yes** on the sender, and the *tls()* option is configured, the communication will be TLS-encrypted. For the sender (syslog-ng PE client), reliable disk-buffering is enabled to prevent data loss.

Receiver configuration (syslog-ng PE server):

```
source  s_syslog_rltp  {
             syslog(
                   ip("127.0.0.1")
                   port("4444")
                   transport(rltp(tls-required(optional)))
                   ip-protocol(4)
                   tls(
                         peer-verify(required-trusted)
                         ca-dir("/var/tmp/client/")
                         key-file("/var/tmp/server/server_priv.key")
                         cert-file("/var/tmp/server/server.crt")
                   )
             );
 };
```

Sender configuration (syslog-ng PE client):

```
destination  d_syslog_rltp  {
             syslog(
                   "127.0.0.1"
                   port("4444")
                   transport(rltp(tls-required(yes)))
                   ip-protocol(4)
                   disk-buffer(  mem-buf-size(200000)  disk-buf-size(2000000)
reliable(yes)  )
                   tls(
                         peer-verify(required-trusted)
                         ca-dir("/var/tmp/server/")
                         key-file("/var/tmp/client/client_priv.key")
                         cert-file("/var/tmp/client/client.crt")
                   )
             );
 };
```

Introduction

Reliable Log Transfer Protocol™ (RLTP™) interacts with flow control and disk buffering to ensure that the loss of log messages is minimized or is prevented completely. This section explains how each loss prevention method contributes to reliability and minimizing log message loss. Flow control, disk buffering, and RLTP™ are explained in detail elsewhere in the document. In this section, we present a high-level overview of all of these mechanisms and highlight considerations such as:

- What best practices exist in various scenarios, how to set key parameters
- When is a log message considered "delivered"
- Under what circumstances can log loss occur

Each of the following sections discusses a different scenario and uses figures to aid comprehension.

NOTE:

Each figure depicts a scenario in which the volume of incoming messages makes it necessary to use all buffers and control windows at maximum capacity.

Important information:

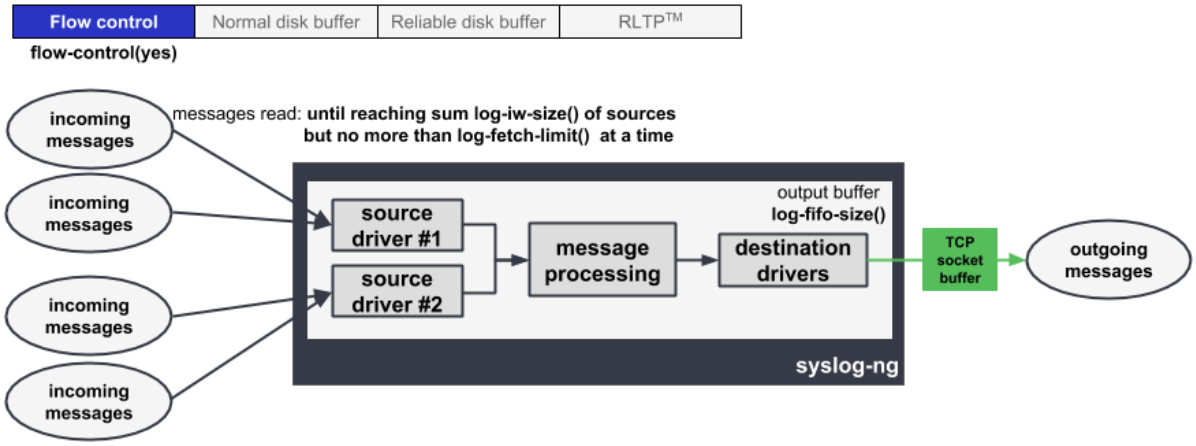
Any of the mechanisms that syslog-ng PE uses to prevent or minimize the loss of log messages only works if the hardware and operating system work normally. When there is an issue with the hardware or operating system that the application and syslog-ng PE run on, log loss may occur. Issues include operating system crash (for example, kernel panic), memory errors, disk errors, power outage, and so on.

Flow control, no disk buffering, no RLTP™

How it works:

log-iw-size() sets a control window that tracks how many messages syslog-ng PE can accept. Every source has its own control window. If the window gets full, syslog-ng PE stops reading messages from the sources until some messages are successfully sent to the destination(s).

Figure 13.1. Flow control, no disk buffering, no RLTP™



How to set key parameters:

Set **`flags(flow-control)`** in the log path.

The output buffer must be large enough to store the incoming messages of every connected source:

`log-fifo-size()` **>** sum of **`log-iw-size()`** of sources connected to this destination

Benefits:

This configuration minimizes the loss of log messages in the following situations:

- *Unreachable destination server(s)*: Only as many incoming log messages are read as can be "delivered". When flow control is used, those messages are considered delivered that have been written to the output buffer. When the output buffer is full, syslog-ng PE stops reading messages from the connected sources. This means that no log messages get lost.

  NOTE:

  In case the application is sending its log messages through a blocking I/O socket, then it is the application that stops sending new log messages and waits

until the previous batch has been delivered. If the application is not sending logs through a blocking I/O socket, then it will keep sending messages (regardless of whether or not the previous batch has been delivered), and this can result in the loss of log messages. For example, it is not possible to apply flow control in the case of a UDP source.

Drawbacks:

While this configuration gives you the fastest processing time, it has some limitations. It does not provide protection against the loss of log messages in the following situations:

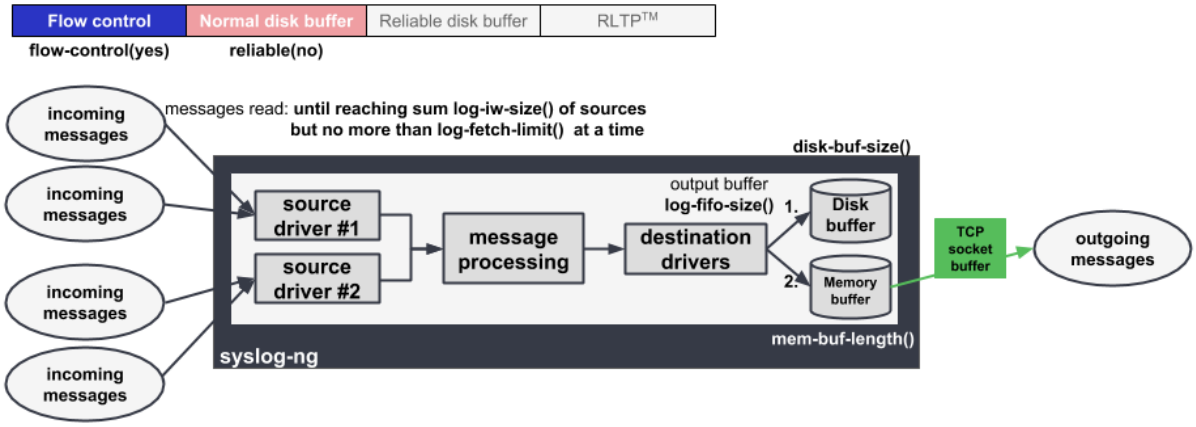
- *TCP error*: In the case of a TCP connection, when messages are sent from the destination drivers to the destination servers, messages are written to the TCP socket. The TCP socket sends an acknowledgement to the destination drivers once it has successfully processed messages. A message is considered "delivered" when no error occurs during the process of writing the data to the socket, and the acknowledgement is received. Note, however, that if something goes wrong after messages have been successfully written to the TCP socket, log messages can still get lost. Also note that TCP errors can occur on both the source and the destination side, and both can cause the loss of log messages.
- *Message loss outside of syslog-ng PE*: Because syslog-ng PE stores only a small number of log messages in the memory, it is possible to lose messages outside of syslog-ng. For example, if the output buffer is full because the server is not reachable, syslog-ng PE will not read the source, meaning that the external application that generates the logs can drop the logs. If you want to minimize the risk, use disk buffering. For details, see the section called “Flow control, normal disk buffering, no RLTP™” and the section called “Flow control, reliable disk buffering, no RLTP™”.
- *Message loss when syslog-ng PE is stopped or restarted*: When syslog-ng is stopped or restarted, the contents of the output buffers are lost. If you want to minimize the risk, use disk buffering. For details, see the section called “Flow control, normal disk buffering, no RLTP™” and the section called “Flow control, reliable disk buffering, no RLTP™”.
- *When syslog-ng PE is not able to operate normally* (for example, when syslog-ng PE crashes due to some unforeseen event): Log messages that were in the output buffer when the issue occurred get lost because those messages are stored in the memory.

Flow control, normal disk buffering, no RLTP™

How it works:

1. syslog-ng PE puts messages into the disk buffer (set via *disk-buf-size()*) when the destination becomes unavailable or when it is not able to process logs as fast as they arrive through the sources.
2. When the disk buffer is full, syslog-ng PE puts messages into the memory buffer (set via *mem-buf-length()*). When the memory buffer gets full too, then syslog-ng PE stops the source (flow-control mechanism).

Figure 13.2. Flow control, normal disk buffering, no RLTP™



How to set key parameters:

Set **`flags(flow-control)`** in the log path.

The memory buffer must be large enough to store the incoming messages of every source:

`mem-buf-length()` **>** sum of **`log-iw-size()`** of sources connected to this destination

Configure disk buffering. For details, see Example 13.1, “Example configuration of normal disk-based buffering”.

Example 13.1. Example configuration of normal disk-based buffering

```
disk-buffer(
    mem-buf-length(20000) # storing 20000 messages in memory, sum log-iw-size
```

```
of sources should be less than 20000 to use flow-control
    disk-buf-size(2147483648) # storing 2 GB of messages on disk
    reliable(no)
)
```

Benefits:

This configuration minimizes the loss of log messages in the following situations:

- *Unreachable destination server(s)*: Only as many incoming log messages are read as can be "delivered". When flow control is used in combination with disk buffering, messages that have been written to the disk buffer and/or the memory buffer are considered delivered. When the memory buffer becomes full, syslog-ng PE stops reading messages from the configured sources. This means that no log messages get lost.
- *Message loss outside of syslog-ng PE*: The greatest advantage of this configuration over when no disk buffering is used at all is that when the **`log-iw-size()`** control window is full, the flow-control mechanism stops reading logs from the sources much later. This is because when it is not possible to send logs directly to the destinations, they are first written to the disk and then the memory buffer. It is only after both the disk buffer and the memory buffer have been filled to their full capacity that the sources are stopped. This enables you to minimize the loss of log messages during peak hours or when the network is temporarily down.
- *Message loss when syslog-ng PE is stopped or restarted*: When syslog-ng is stopped or restarted, the contents of the memory buffer and the disk buffer are flushed to disk, meaning that no log loss occurs.

  NOTE:

  In rare cases, the buffers stored on the disk can become corrupted, in which case syslog-ng PE may not able to process all the logs stored in the disk buffer.

Drawbacks:

One drawback of using disk buffering is that the processing of log messages by syslog-ng PE is slower.

This configuration does not provide protection against the loss of log messages in the following situations:

- *TCP error*: In the case of a TCP connection, when messages are sent from the destination drivers to the destination servers, messages are written to the TCP socket. The TCP socket sends an acknowledgement to the destination drivers once it has successfully processed messages. A message is considered "delivered" when no error occurs during the process of writing the data to the socket, and the acknowledgement is received. Note, however, that if something goes wrong after

messages have been successfully written to the TCP socket, log messages can still get lost. Also note that TCP errors can occur on both the source and the destination side, and both can cause the loss of log messages.

- *When syslog-ng PE is not able to operate normally* (for example, when syslog-ng PE crashes due to some unforeseen event): Log messages that were in the output buffer when the issue occurred get lost because those messages are stored in the memory.

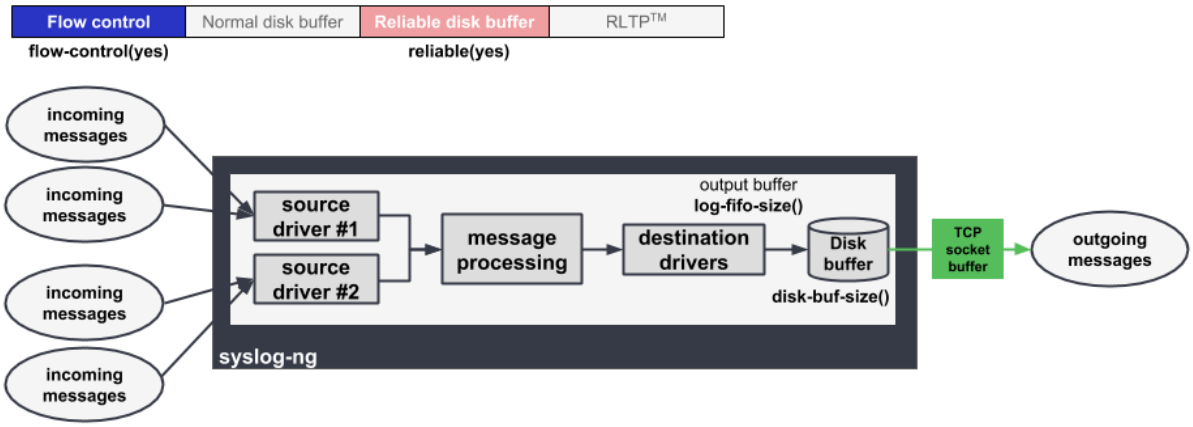
Flow control, reliable disk buffering, no RLTP™

How it works:

syslog-ng puts messages into the disk buffer, until disk-buffer size reaches *disk-buf-size()*. Above that size, flow control is triggered. syslog-ng PE completely stops reading incoming messages from the source, making the control window (governed by *log_iw_size()*) fill up and blocking the sources.

In this configuration, log messages are stored on the disk (and not in the memory), which increases reliability.

Figure 13.3. Flow control, reliable disk buffering, no RLTP™



How to set key parameters:

Set `flags(flow-control)` in the log path.

Configure disk buffering. For details, see Example 13.2, “Example configuration of reliable disk-based buffering”.

Example 13.2. Example configuration of reliable disk-based buffering

```
disk-buffer(
       mem-buf-size(10485760) # storing 10 MB of messages in memory and
on disk
       disk-buf-size(2147483648) # storing 2 GB of messages only on disk
       reliable(yes)
 )
```

Benefits:

This configuration minimizes the loss of log messages in the following situations:

- *Unreachable destination server(s)*: Only as many incoming log messages are read as can be "delivered". When flow control is used in combination with disk buffering, those messages are considered delivered that have been written to the disk buffer. As soon as the disk buffer is full, syslog-ng PE stops reading messages. This means that no log messages get lost.
- *Message loss outside of syslog-ng PE*: One of the advantages of this configuration over when no disk buffering is used at all is that when the **`log-iw-size()`** control window is full, the flow-control mechanism stops reading logs from the sources much later. This is because when it is not possible to send logs directly to the destinations, they are written to the disk. It is only after the disk buffer has been filled to its full capacity that the sources are stopped. This enables you to minimize the loss of log messages during peak hours or when the network is temporarily down.
- *Message loss when syslog-ng PE is stopped or restarted*: When syslog-ng is stopped or restarted, the contents of the disk buffer do not get lost, greatly increasing reliability.

  Also note that the memory buffer is only used as a cache in this configuration. Any data stored in the memory has already been written to the disk buffer, which, again, results in more reliability.

  > NOTE:
  >
  > In rare scenarios, the buffers stored on the disk can become corrupted, in which case syslog-ng PE may not able to process all the logs stored in the disk buffer.

- *When syslog-ng PE is not able to operate normally* (for example, when syslog-ng PE crashes due to some unforeseen event): No messages get lost because the disk buffer is persistent and when the disk buffer is full, syslog-ng PE stops reading messages from the sources. When syslog-ng PE is restarted after a crash, it automatically recovers any unsent messages from the disk buffer and the output buffer. After the restart, syslog-ng PE sends the saved messages to the destination.

Drawbacks:

One drawback of using reliable disk buffering is that the processing of log messages by syslog-ng PE is slower than when messages are stored in the output buffer only, or when using normal disk buffering.

This configuration does not provide protection against the loss of log messages in the following situations:

- *TCP error*: In the case of a TCP connection, when messages are sent from the destination drivers to the destination servers, messages are written to the TCP socket. The TCP socket sends an acknowledgement to the destination drivers once it has successfully processed messages. A message is considered "delivered" when no error occurs during the process of writing the data to the socket, and the acknowledgement is received. Note, however, that if something goes wrong after messages have been successfully written to the TCP socket, log messages can still

get lost. Also note that TCP errors can occur on both the source and the destination side, and both can cause the loss of log messages.

Flow control, reliable disk buffering, RLTP™

How it works:

NOTE:

The example presented here is set in a client-relay-server scenario.

1. The sender sends messages in batches (set via `flush-lines()`).
2. The relay writes messages to the disk buffer.
3. Once messages have been written to the disk buffer, the relay returns an acknowledgement to the client.
4. The relay sends messages to the server in batches (set via `flush-lines()`).
5. When the server has successfully received and processed the messages in the batch, it sends an acknowledgement of the processed messages to the relay.

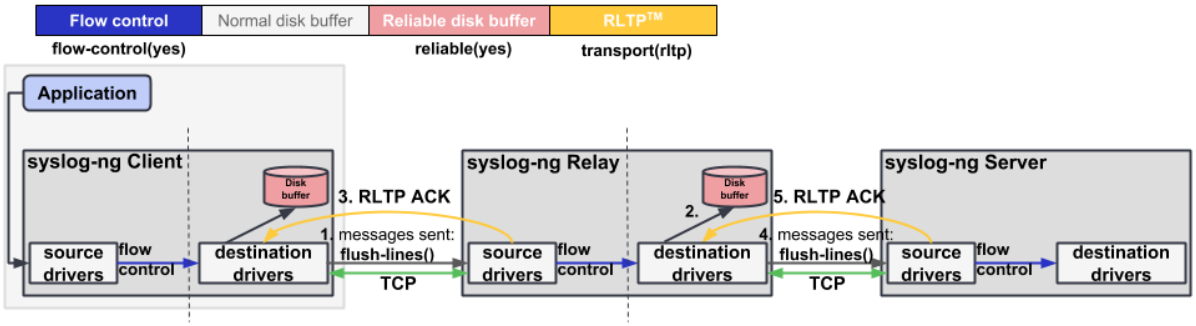
   It is only at this point that the relay removes log messages from the disk buffer because this is when logs are considered "delivered" to the server.

   After receiving the acknowledgement, the sender sends another batch of messages.

This configuration gives you the greatest degree of protection against log message loss. RLTP™ provides acknowledgement about the successful processing of log messages at the level of the application layer. Even if the reception of log messages has been acknowledged by TCP at the transport layer, log messages are considered delievered only when the syslog-ng PE application has received an acknowledgement from the other syslog-ng PE instance about the successful delivery of log messages.

This mechanism guarantees that log messages are not lost between the client and the relay, or between the relay and the server, or on the relay itself. To minimize the risk of message loss on the client or the server, use flow control and reliable disk buffering.

Figure 13.4. Flow control, reliable disk buffering, RLTP™



How to set key parameters:

Set **`flags(flow-control)`** in the log path.

Configure disk buffering. For details, see Example 13.3, "Example configuration of reliable disk-based buffering".

Example 13.3. Example configuration of reliable disk-based buffering

```
disk-buffer(
      mem-buf-size(10485760) # storing 10 MB of messages in memory and
on disk
      disk-buf-size(2147483648) # storing 2 GB of messages only in disk
      reliable(yes)
 )
```

Enable RLTP™ by setting **`transport(rltp)`**. For details, see the section called "RLTP™ options".

Benefits:

- *Unreachable destination server(s)*: Only as many incoming log messages are read as can be "delivered". When flow control is used in combination with reliable disk buffering and RLTP™, those messages are considered delivered by the very first source driver that have been written to the disk buffer. syslog-ng PE will not read new messages until the previous batch has been written to the disk buffer.
- *TCP error*: In the case of a TCP connection, when messages are sent from the destination drivers to the destination servers, messages are written to the TCP socket. The TCP socket sends an acknowledgement to the destination drivers once it has successfully processed messages. However, while the acknowledge messages sent by the TCP socket implement flow control at the transport layer, RLTP™ introduces flow control at the application layer. This means that log messages are only considered delivered, when the RLTP™ acknowledge message is returned at the level of the syslog-ng application. That is to say, when a TCP error occurs, messages that have been written to the disk buffer do not get lost.
- *Message loss outside of syslog-ng PE*: One of the advantages of this configuration over when no disk buffering is used at all is that when the **`log-iw-size()`** control window is full, the flow-control mechanism stops reading logs from the sources much later. This is because when it is not possible to send logs directly to the destinations, they are written to the disk. It is only after the disk buffer has been filled to its full capacity that the sources are stopped. This enables you to minimize the loss of log messages during peak hours or when the network is temporarily down.
- *Message loss when syslog-ng PE is stopped or restarted*: When syslog-ng is stopped or restarted, the contents of the disk buffer do not get lost, greatly increasing reliability.

Also note that the memory buffer is only used as a cache in this configuration. Any data stored in the memory has already been written to the disk buffer, which, again, results in more reliability.

> NOTE:
>
> In rare scenarios, the buffers stored on the disk can become corrupted, in which case syslog-ng PE may not able to process all the logs stored in the disk buffer.

- *When syslog-ng PE is not able to operate normally* (for example, when syslog-ng PE crashes due to some unforeseen event): No messages get lost because the disk buffer is persistent and when the disk buffer is full, syslog-ng PE stops reading messages from the sources. When syslog-ng PE is restarted after a crash, it automatically recovers any unsent messages from the disk buffer and the output buffer. After the restart, syslog-ng PE sends the saved messages to the destination.

Drawbacks:

This configuration results in the slowest processing time out of all the options described in this chapter.

Deciding which loss prevention mechanism to apply

Choosing the ideal configuration for your environment may not always be a starightforward decision. Depending on your use case, it is worth considering which outcome is more desirable (with the following points representing the two opposite ends of the spectrum):

- an application that does not slow down or stop — at the price of losing logs
- no log messages get lost — at the price of a slower application or an application that stops (temporarily)

TIP:

If your application sends its logs through a blocking I/O socket and you prefer not to slow down or stop the application when log messages are arriving in volumes greater than syslog-ng PE is able to process, then consider turning flow control off on the client side. This way, you will not be using the whole application-client-server chain at full capacity, and yet still be able to spot the loss of application log messages at the beginning of the chain already, in the internal logs of the client.

This chapter explains the methods that you can use to customize, reformat, and modify log messages using syslog-ng Premium Edition.

- the section called "Customizing message format" explains how to use templates and macros to change the format of log messages, or the names of logfiles and database tables.
- the section called "Modifying messages" describes how to use rewrite rules to search and replace certain parts of the message content.
- the section called "Regular expressions" lists the different types of regular expressions that can be used in various syslog-ng PE objects like filters and rewrite rules.

Customizing message format

The following sections describe how to customize the names of logfiles, and also how to use templates, macros, and template functions.

- the section called "Formatting messages, filenames, directories, and tablenames" explains how macros work.
- the section called "Modifying messages" describes how to use macros and templates to format log messages or change the names of logfiles and database tables.
- the section called "Macros of syslog-ng PE" lists the different types of macros available in syslog-ng PE.
- the section called "Using template functions" explains what template functions are and how to use them.
- the section called "Template functions of syslog-ng PE" lists the template functions available in syslog-ng PE.

Formatting messages, filenames, directories, and tablenames

The syslog-ng PE 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:

If a message uses the IETF-syslog format (RFC5424), only the text of the message can be customized (that is, the $MESSAGE part of the log), the structure of the header is fixed.

- For details on using templates and macros, see the section called "Templates and macros".
- For a list and description of the macros available in syslog-ng PE, see the section called "Macros of syslog-ng PE".
- For details on using custom macros created with CSV parsers and pattern databases, see Chapter 15, *Parsing and segmenting structured messages* and the section called "Using parser results in filters and templates", respectively.

Templates and macros

The syslog-ng PE application allows you to define message templates, and reference them from every object that can use a template. Templates can include strings, macros (for example date, the hostname, and so on), and template functions. For example, you can use templates to create standard message formats or filenames. For a list of macros available in syslog-ng Premium Edition, see the section called "Macros of syslog-ng PE". For the macros of the syslog-ng Agent for Windows application, see *Administration Guide for syslog-ng Agent for Windows*. Fields from the structured data (SD) part of messages using the new IETF-syslog standard can also be used as macros.

Declaration:

```
template <template-name> {
       template("<template-expression>") <template-escape(yes)>;
 };
```

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 backslash 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 **$** sign, just like in Bourne compatible shells. Although using braces around macro names is not mandatory, and the **"$MSG"** and **"${MSG}"** formats are equivalent, using the **"${MSG}"** format is recommended for clarity.

To use a literal `$` character in a template, you have to escape it. In syslog-ng PE versions 4.0-4.2, use a backslash (`\$`). In version 5.0 and later, use `$$`.

NOTE:

To use a literal `@` character in a template, use `@@`.

Default values for macros can also be specified by appending the `:-` characters and the default value of the macro. If a message does not contain the field referred to by the macro, or it is empty, the default value will be used when expanding the macro. For example, if a message does not contain a hostname, the following macro can specify a default hostname.

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

NOTE:

For the macros of the syslog-ng Agent for Windows application, see *Administration Guide for syslog-ng Agent for Windows*.

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. For details, see the section called "sql() destination options".

Example 14.1. Using templates and macros

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) );
                };
```

The following file destination uses macros to daily create separate logfiles for every client host.

```
destination  d_file  {
             file("/var/log/${YEAR}.${MONTH}.${DAY}/${HOST}.log");
 };
```

NOTE:

Macros can be used to format messages, and also in the name of destination files or database tables. 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.

Date-related macros

The macros related to the date of the message (for example: *${ISODATE}*, *${HOUR}*, and so on) have two further variants each:

- **S_** prefix, for example, *${S_DATE}*: The *${S_DATE}* macro represents the date found in the log message, that is, when the message was sent by the original application.

  CAUTION:

  To use the S_ macros, the *keep-timestamp()* option must be enabled (this is the default behavior of syslog-ng PE).

- **R_** prefix, for example, *${R_DATE}*: *${R_DATE}* is the date when syslog-ng PE has received the message.

The *${DATE}* macro equals the *${S_DATE}* macro.

The values of the date-related macros are calculated using the original timezone information of the message. To convert it to a different timezone, use the *time-zone()* option. You can set the *time-zone()* option as a global option, or per destination. For sources, it applies only if the original message does not contain timezone information. Converting the timezone changes the values of the following date-related macros (macros *MSEC* and *USEC* are not changed):

- *AMPM*

- *DATE*
- *DAY*
- *FULLDATE*
- *HOUR*
- *HOUR12*
- *ISODATE*
- *MIN*
- *MONTH*
- *MONTH_ABBREV*
- *MONTH_NAME*
- *MONTH_WEEK*
- *SEC*
- *STAMP*
- *TZ*
- *TZOFFSET*
- *UNIXTIME*
- *WEEK*
- *WEEK_DAY*
- *WEEK_DAY_ABBREV*
- *WEEK_DAY_NAME*
- *YEAR*
- *YEAR_DAY*

Hard vs. soft macros

Hard macros contain data that is directly derived from the log message, for example, the ${MONTH} macro derives its value from the timestamp. Hard macros are read-only. Soft macros (sometimes also called name-value pairs) are either built-in macros automatically generated from the log message (for example, ${HOST}), or custom user-created macros

generated by using the syslog-ng pattern database or a CSV-parser. In contrast to hard macros, soft macros are writable and can be modified within syslog-ng PE, for example, using rewrite rules.

Hard and soft macros are rather similar and often treated as equivalent. Macros are most commonly used in filters and templates, which does not modify the value of the macro, so both soft and hard macros can be used. However, it is not possible to change the values of hard macros in rewrite rules or via any other means.

The following macros in syslog-ng PE are hard macros and cannot be modified: *BSDTAG*, *CONTEXT_ID*, *DATE*, *DAY*, *FACILITY_NUM*, *FACILITY*, *FULLDATE*, *HOUR*, *ISODATE*, *LEVEL_NUM*, *LEVEL*, *MIN*, *MONTH_ABBREV*, *MONTH_NAME*, *MONTH*, *MONTH_WEEK*, *PRIORITY*, *PRI*, *RCPTID*, *SDATA*, *SEC*, *SEQNUM*, *SOURCEIP*, *STAMP*, *TAG*, *TAGS*, *TZOFFSET*, *TZ*, *UNIXTIME*, *WEEK_DAY_ABBREV*, *WEEK_DAY_NAME*, *WEEK_DAY*, *WEEK*, *YEAR_DAY*, *YEAR*.

The following macros can be modified: *FULLHOST_FROM*, *FULLHOST*, *HOST_FROM*, *HOST*, *LEGACY_MSGHDR*, *MESSAGE*, *MSG*,*MSGID*, *MSGONLY*, *PID*, *PROGRAM*, *SOURCE*. Custom values created using rewrite rules or parsers can be modified as well, just like stored matches of regular expressions ($0 ... $255).

Macros of syslog-ng PE

The following macros are available in syslog-ng PE.

CAUTION:

These macros are available when syslog-ng PE successfully parses the incoming message as a syslog message, or you use some other parsing method and map the parsed values to these macros.

If you are using the *flags(no-parse)* option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since *flags(no-parse)* disables message parsing, it interferes with other flags, for example, disables *flags(no-multi-line)*.

AMPM

Description: Typically used together with the *${HOUR12}* macro, *${AMPM}* returns the period of the day: AM for hours before mid day and PM for hours after mid day. In reference to a 24-hour clock format, AM is between 00:00-12:00 and PM is between 12:00-24:00. 12AM is midnight. Available in syslog-ng PE 3.2 and later.

APP_NAME

Description: An alias for the *APPLICATION_NAME* macro.

APPLICATION_NAME

Description:

- At event container: Name of the application the message came from
- At file as the name of creator (default value): syslog-ng-agent

BSDDATE, R_BSDDATE, S_BSDDATE

Description: Date of the message in BSD 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`. If possible, it is recommended to use *ISODATE* for timestamping.

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), and so on.

Custom macros

Description: CSV parsers and pattern databases can also define macros from the content of the messages, for example, a pattern database rule can extract the username from a login message and create a macro that references the username. For details on using custom macros created with CSV parsers and pattern databases, see Chapter 15, *Parsing and segmenting structured messages* and the section called "Using parser results in filters and templates", respectively.

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.

FILE_FACILITY

Description: The facility that sent the message.

FILE_LEVEL

Description: Importance level of the message represented as a number: 6 - Success, 5 - Informational, 4- Warning, or 3 - Error).

FILE_MESSAGE

Description: The content of the message.

FILE_MSG

Description: The content of the message. This is an alias of the *FILE_MESSAGE* macro.

FILE_NAME

Description: Name of the log file (including its path) from where the syslog-ng PE received 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 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.
- If the *keep-hostname()* option is disabled (**keep-hostname(no)**), the value of the $FULLHOST macro will be the DNS hostname of the host that sent the message to syslog-ng PE (that is, the DNS hostname of the last hop). In this case the $FULLHOST and $FULLHOST_FROM macros will have the same value.
- If the *keep-hostname()* option is enabled (**keep-hostname(yes)**), the value of the $FULLHOST macro will be the hostname retrieved from the log message. That way the name of the original sender host can be used, even if there are log relays between the sender and the server.

  NOTE:

  The *use-dns()*, *use-fqdn()*, *normalize-hostnames()*, and *dns-cache()* options will have no effect if the *keep-hostname()* option is enabled (**keep-hostname(yes)**) and the message contains a hostname.

For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

FULLHOST_FROM

Description: The 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.

The syslog-ng PE application uses the following procedure to determine the value of the *$FULLHOST_FROM* macro:

1. The syslog-ng PE application takes the IP address of the host sending the message.
2. If the *use-dns()* option is enabled, syslog-ng PE attempts to resolve the IP address to a hostname. If it succeeds, the returned hostname will be the value of the *$FULLHOST_FROM* macro. This value will be the FQDN of the host if the *use-fqdn()* option is enabled, but only the hostname if *use-fqdn()* is disabled.
3. If the *use-dns()* option is disabled, or the address resolution fails, the *${FULLHOST_FROM}* macro will return the IP address of the sender host.

For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

HOUR, R_HOUR, S_HOUR

Description: The hour of day the message was sent.

HOUR12, R_HOUR12, S_HOUR12

Description: The hour of day the message was sent in 12-hour clock format. See also the *${AMPM}* macro. 12AM is midnight. Available in syslog-ng PE 3.2 and later.

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.
- If the *keep-hostname()* option is disabled (**keep-hostname(no)**), the value of the $HOST macro will be the DNS hostname of the host that sent the message to syslog-ng PE (that is, the DNS hostname of the last hop). In this case the $HOST and $HOST_FROM macros will have the same value.
- If the *keep-hostname()* option is enabled (**keep-hostname(yes)**), the value of the $HOST macro will be the hostname retrieved from the log message. That way the

name of the original sender host can be used, even if there are log relays between the sender and the server.

> NOTE:
>
> The *use-dns()*, *use-fqdn()*, *normalize-hostnames()*, and *dns-cache()* options will have no effect if the *keep-hostname()* option is enabled (**keep-hostname(yes)**) and the message contains a hostname.

For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

HOST_FROM

Description: The 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.

The syslog-ng PE application uses the following procedure to determine the value of the *$HOST_FROM* macro:

1. The syslog-ng PE application takes the IP address of the host sending the message.
2. If the *use-dns()* option is enabled, syslog-ng PE attempts to resolve the IP address to a hostname. If it succeeds, the returned hostname will be the value of the *$HOST_FROM* macro. This value will be the FQDN of the host if the *use-fqdn()* option is enabled, but only the hostname if *use-fqdn()* is disabled.
3. If the *use-dns()* option is disabled, or the address resolution fails, the *${HOST_FROM}* macro will return the IP address of the sender host.

For details on using name resolution in syslog-ng PE, see the section called "Using name resolution in syslog-ng".

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. For the textual representation of this value, use the *${LEVEL}* macro. See the section called "PRIORITY or LEVEL" for details.

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.

MONTHNAME, R_MONTHNAME, S_MONTHNAME

Description: The English name of the month the message was sent, abbreviated to three characters (for example Jan, Feb, and so on).

MSEC, R_MSEC, S_MSEC

Description: The millisecond the message was sent.

Available in syslog-ng PE version 4 F2 and later.

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.

If you are using the *flags(no-parse)* option, then syslog message parsing is completely disabled, and the entire incoming message is treated as the ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since *flags(no-parse)* disables message parsing, it interferes with other flags, for example, disables *flags(no-multi-line)*.

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.

MSGID

Description: A string specifying the type of the message in IETF-syslog (RFC5424-formatted) messages. For example, a firewall might use the ${MSGID} "TCPIN" for incoming TCP traffic and the ${MSGID} "TCPOUT" for outgoing TCP traffic. By default, syslog-ng PE does not specify this value, but uses a dash (-) character instead. If an incoming message includes the ${MSGID} value, it is retained and relayed without modification.

MSGONLY

Description: Message contents without the program name or pid.

OSUPTIME

Description: The time elapsed since the computer running syslog-ng PE has booted. The value of this macro is an integer containing the time in 1/100th of the second. Note that currently syslog-ng PE can access this data only on Linux platforms. On other platforms the macro contains the time elapsed since syslog-ng PE was started.

Note that syslog-ng PE evaluates the macro every time it is processed, so even if you use the same macro for the same message, its value can be different. For example, if you use it in a filter and in a destination filename, their values will be different even for the same message.

Available in syslog-ng PE version 6.0.5 and later.

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*. For the textual representation of this value, use the *${LEVEL}* macro. See the section called "PRIORITY or LEVEL" for details.

PROCESS_ID

Description: PID of the application the message came from.

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.

RCPTID

Description: This is disabled by default due to performance issues. To enable it, add the next option to the global options: **`use-uniqid(yes);`**. For details, see also the section called "use-uniqid()". A unique ID for messages generated at reception time on the receiving host. It facilitates defining relationships between messages that are potentially distributed to different files on the same host, or different hosts.

Example 14.2. Using ${RCPTID} macro

Using the following template statement in the configuration: **`template demo_template { template("${DATE} ${HOST} ${PROG}: ${MSG} ID:${RCPTID}\n"); };`** the outgoing message will be the following: **`<133>Feb 25 14:09:07 webserver syslogd: restart. ID:1.`**

SDATA, .SDATA.SDID.SDNAME, STRUCTURED_DATA

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.Available only in syslog-ng Premium Edition 4.0 and later.

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 14.3. 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 *${SEQNUM}* macro contains a sequence number for the log message. The value of the macro depends on the scenario, and can be one of the following:

- If syslog-ng PE receives a message via the IETF-syslog protocol that includes a sequence ID, this ID is automatically available in the *${SEQNUM}* macro.
- If the message is a Cisco IOS log message using the extended timestamp format, then syslog-ng PE stores the sequence number from the message in this macro. If you forward this message the IETF-syslog protocol, syslog-ng PE includes the sequence number received from the Cisco device in the **${.SDATA.meta.sequenceId}** part of the message.

  NOTE:

  To enable sequence numbering of log messages on Cisco devices, use the following command on the device (available in IOS 10.0 and later): **service sequence-numbers**. For details, see the manual of your Cisco device.

- For locally generated messages (that is, for messages that are received from a local source, and not from the network), syslog-ng PE calculates a sequence number when sending the message to a destination (it is not calculated for relayed messages).
  - The sequence number is not global, but per-destination. Essentially, it counts the number of messages sent to the destination.
  - This sequence number increases by one for every message sent to the destination, and is not lost even when syslog-ng PE is reloaded or restarted.
  - This sequence number is added to every message that uses the IETF-syslog protocol (**${.SDATA.meta.sequenceId}**), and can be added to BSD-syslog messages using the *${SEQNUM}* macro.

NOTE:

If you need a sequence number for every log message that syslog-ng PE receives, use the RCPTID macro.

SOURCE

Description: The identifier of the source statement in the syslog-ng PE configuration file that received the message. For example, if syslog-ng PE received the log message from the **`source s_local { internal(); };`** source statement, the value of the ${SOURCE} macro is **`s_local`**. This macro is mainly useful for debugging and troubleshooting purposes.

SOURCEIP

Description: IP address of the host that sent the message to syslog-ng. (That is, 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.

SYSUPTIME

Description: The time elapsed since the syslog-ng PE instance was started (that is, the uptime of the syslog-ng PE process). The value of this macro is an integer containing the time in 1/100th of the second. For the uptime of the host running syslog-ng PE see the section called "OSUPTIME".

Note that syslog-ng PE evaluates the macro every time it is processed, so even if you use the same macro for the same message, its value can be different. For example, if you use it in a filter and in a destination filename, their values will be different even for the same message.

Available in syslog-ng PE version 4 F1 and later.

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. Available only in syslog-ng Premium Edition 3.2 and later.

> 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: An alias of the ${TZOFFSET} macro.

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`.

UNIQID

Description: A globally unique ID generated from the HOSTID and the RCPTID in the format of HOSTID@RCPTID. For details, see the section called "use-uniqid()" and the section called "RCPTID".

Available in syslog-ng PE version 5 F2 and later.

USEC, R_USEC, S_USEC

Description: The microsecond the message was sent.

Available in syslog-ng PE version 4 F2 and later.

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.

Using template functions

A template function is a transformation: it modifies the way macros or name-value pairs are expanded. Template functions can be used in template definitions, or when macros are used in the configuration of syslog-ng PE. Template functions use the following syntax:

```
$(function-name parameter1 parameter2 parameter3 ...)
```

For example, the *`$(echo)`* template function simply returns the value of the macro it receives as a parameter, thus **`$(echo ${HOST})`** is equivalent to *`${HOST}`*.

The parameters of template functions are separated by a whitespace character. If you want to use a longer string or multiple macros as a single parameter, enclose the parameter in double-quotes or apostrophes. For example:

```
$(echo "${HOST} ${PROGRAM} ${PID}")
```

Template functions can be nested into each other, so the parameter of a template function can be another template function, like:

```
$(echo $(echo ${HOST}))
```

For details on using template functions, see the descriptions of the individual template functions in the section called “Template functions of syslog-ng PE”.

Template functions of syslog-ng PE

The following template functions are available in syslog-ng PE.

echo

Syntax:

```
$(echo argument)
```

Description: Returns the value of its argument. Using **`$(echo ${HOST})`** is equivalent to *`${HOST}`*.

format-cef-extension

syslog-ng PE version 5 F6 includes a new template function (*`format-cef-extension`*) to format name-value pairs as ArcSight Common Event Format extensions. Note that the template function only formats the selected name-value pairs, it does not provide any mapping. There is no special support for creating the prefix part of a Common Event Format (CEF) message. Note that the order of the elements is random. For details on the CEF extension escaping rules format, see the ArcSight Common Event Format.

You can use the value-pairs that syslog-ng PE stores about the log message as CEF fields. Using value-pairs, you can:

- select which value-pairs to use as CEF fields,
- add custom value-pairs as CEF fields,
- rename value-pairs, and so on.

For details, see the section called "Structuring macros, metadata, and other value-pairs". Note that the syntax of **`format-*`** template functions is different from the syntax of *`value-pairs()`*: these template functions use a syntax similar to command lines.

Using the *`format-cef-extension`* template function, has the following prerequisites:

- Load the the **`cef`** module in your configuration:

  ```
  @module cef
  ```

- Set the *`on-error`* global option to *`drop-property`*, otherwise if the name of a name-value pair includes an invalid character, syslog-ng PE drops the entire message. (Key name in CEF extensions can contain only the A-Z, a-z and 0-9 characters.)

  ```
  options  {
       on-error("drop-property");
   };
  ```

- The log messages must be encoded in UTF-8. Use the *`encoding()`* option or the *`validate-utf8`* flag in the message source.

Example 14.4. Using the *`format-cef-extension`* template function

The following example selects every available information about the log message, except for the date-related macros (**R_*** and **S_***), selects the **.SDATA.meta.sequenceId** macro, and defines a new value-pair called **MSGHDR** that contains the program name and PID of the application that sent the log message (since you will use the template-function in a template, you must escape the double-quotes).

```
$(format-cef-extension --scope syslog,all_macros,selected_macros \
  --exclude R_* --exclude S_* --key .SDATA.meta.sequenceId \
  --pair MSGHDR=\"$PROGRAM[$PID]: \")
```

The following example selects every value-pair that has a name beginning with **.cef.**, but removes the **.cef.** prefix from the key names.

```
template("$(format-cef-extension --subkeys .cef.)\n")
```

The following example shows how to use this template function to store log messages in CEF format:

```
destination d_cef_extension {
    file("/var/log/messages.cef" template("${ISODATE} ${HOST} $(format-cef-
extension --scope selected_macros --scope nv_pairs)\n"));
 };
```

format-json

Syntax:

```
$(format-json parameters)
```

Description: The *format-json* template function receives value-pairs as parameters and converts them into JavaScript Object Notation (JSON) format. Including the template function in a message template allows you to store selected information about a log message (that is, its content, macros, or other metadata) in JSON format. Note that the input log message does not have to be in JSON format to use **format-json**, you can reformat any incoming message as JSON.

You can use the value-pairs that syslog-ng PE stores about the log message as JSON fields. Using value-pairs, you can:

- select which value-pairs to use as JSON fields,
- add custom value-pairs as JSON fields,
- rename value-pairs, and so on.

For details, see the section called "Structuring macros, metadata, and other value-pairs". Note that the syntax of *format-json* is different from the syntax of *value-pairs()*: *format-json* uses a syntax similar to command lines.

NOTE:

By default, syslog-ng PE handles every message field as a string. For details on how to send selected fields as other types of data (for example, handle the PID as a number), see the section called "Specifying data types in value-pairs".

Example 14.5. Using the format-json template function

The following example selects every available information about the log message, except for the date-related macros (**R_*** and **S_***), selects the **.SDATA.meta.sequenceId** macro, and defines a new value-pair called **MSGHDR** that contains the program name and PID of the application that sent the log message (since you will use the template-function in a template, you must escape the double-quotes).

```
$(format-json --scope syslog,all_macros,selected_macros \
  --exclude R_* --exclude S_* --key .SDATA.meta.sequenceId \
  --pair MSGHDR=\"$PROGRAM[$PID]: \")
```

The following example shows how to use this template function to store log messages in JSON format:

```
destination d_json {
    file("/var/log/messages.json" template("$(format_json --scope selected_
macros --scope nv_pairs)\n"));
 };
```

NOTE:

In case of syslog-ng macros starting with a dot (for example "**.SDATA.meta.sequenceID**") an empty key name is added at the top level of the JSON structure. You can work around this by adding **--shift 1** as a parameter to the template function. For example in case of "**.SDATA.meta.sequenceID**", an empty key name is added at the top level of the JSON structure:

```
{"":
    {"SDATA" :
        {"meta" :
            {"sequenceID": "123"}
        }
    }
}
```

format-welf

This template function converts value-pairs into the WebTrends Enhanced Log file Format (WELF). The WELF format is a comma-separated list of **name=value** elements. Note that the order of the elements is random. If the value contains whitespace, it is enclosed in double-quotes, for example, **name="value"**. For details on the WELF format, see https://www3.trustwave.com/support/kb/article.aspx?id=10899.

To select which value-pairs to convert, use the command-line syntax of the *value-pairs()* option. For details on selecting value-pairs, see the section called "value-pairs()".

Example 14.6. Using the *format-welf()* template function

The following example selects every available information about the log message, except for the date-related macros (**R_*** and **S_***), selects the **.SDATA.meta.sequenceId** macro, and defines a new value-pair called **MSGHDR** that contains the program name and PID of the application that sent the log message (since you will use the template-function in a template, you must escape the double-quotes).

```
$(format-welf --scope syslog,all_macros,selected_macros \
  --exclude R_* --exclude S_* --key .SDATA.meta.sequenceId \
  --pair MSGHDR=\"$PROGRAM[$PID]: \")
```

The following example shows how to use this template function to store log messages in WELF format:

```
destination d_welf {
    file("/var/log/messages.welf" template("$(format-welf --scope selected_
macros --scope nv_pairs)\n"));
 };
```

grep

Syntax:

```
$(grep condition value-to-select)
```

Description: The *grep* template function is useful when using a pattern database to correlate related log messages. The *grep* template function can be used to filter the messages of the same context when the index of the particular message is not known.

Example 14.7. Using the grep template function

The following example selects the message of the context that has a **username**

name-value pair with the **root** value, and returns the value of the **auth_method** name-value pair.

```
$(grep ("${username}" == "root") ${auth_method})
```

It is possible to specify multiple name-value pairs as parameters, separated with commas. If multiple messages match the condition of *grep*, these will be returned also separated by commas. This can be used for example to collect the e-mail recipients from postfix messages.

hash

Syntax:

```
$(<method> [opts] $arg1 $arg2 $arg3...)
```

Options:

```
--length N, -l N
```

Truncate the hash to the first N characters.

Description: Calculates a hash of the string or macro received as argument using the specified hashing method. If you specify multiple arguments, effectively you receive the hash of the first argument salted with the subsequent arguments.

<method> can be one of md5, md4, sha1, sha256, sha512 and "hash", which is equivalent to sha256. Macros are expected as arguments, and they are concatenated without the use of additional characters.

This template function can be used for anonymizing sensitive parts of the log message (for example username) that were parsed out using PatternDB before storing or forwarding the message. This way, the ability of correlating messages along this value is retained.

Also, using this template, quasi-unique IDs can be generated for data, using the *--length* option. This way, IDs will be shorter than a regular hash, but there is a very small possibility of them not being as unique as a non-truncated hash.

Example 14.8. Using the $(hash) template function

The following example calculates the SHA1 hash of the hostname of the message:

```
$(sha1 $HOST)
```

The following example calculates the SHA256 hash of the hostname, using the **salted** string to salt the result:

```
$(sha1 $HOST salted)
```

To use shorter hashes, set the *--length*:

```
$(sha1 --length 6 $HOST)
```

To replace the hostname with its hash, use a rewrite rule:

```
rewrite r_rewrite_hostname{set("$(sha1 $HOST)", value("HOST"));};
```

Example 14.9. Anonymizing IP addresses

The following example replaces every IPv4 address in the MESSAGE part with its SHA-1 hash:

```
rewrite  pseudonymize_ip_addresses_in_message  {
    subst  (
       "((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])[.]){3}([0-9]|[1-9]
[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]))",
       "$(sha1  $0)",
       value("MSG"),
       flags(global)
    );
 };
```

if

Syntax:

```
$(if (<condition>) <true template> <false template>)
```

Description: Returns the value of the **<true template>** parameter if the **<condition>** is true. If the **<condition>** is false, the value of **<false template>** is returned.

Example 14.10. Using pattern databases and the if template function

The following example returns **violation** if the **username** name-value pair of a message processed with pattern database is **root**, and **system** otherwise.

```
$(if ('${username}' == 'root') 'violation' 'system')
```

This can be used to set the class of a message in pattern database rules based on the condition.

```
<value name="username">$(if ("${username}" == "root") "violation"
"system")</value>
```

Since template functions can be embedded into each other, it is possible to use another template function as the template of the first one. For example, the following expression returns **root** if the username is **root**, **admin** if the username is **joe**, and **normal user** otherwise.

```
<value name="username">    $(if ("${username}" == "root")
        "root"
        $(if ("${username}" == "joe") "admin" "normal user")
        "normal user")</value>
```

indent-multi-line

Syntax:

```
$(indent-multi-line parameter)
```

Description: This template function makes it possible to write multi-line log messages into a file. The first line is written like a regular message, subsequent lines are indented with a tab, in compliance with RFC822.

Example 14.11. Using the indent-multi-line template function

The following example writes multi-line messages into a text file.

```
destination  d_file  {
             file  ("/var/log/messages"
                          template("${ISODATE}  ${HOST}  $(indent-multi-line
${MESSAGE})\n")  );
 };
```

ipv4-to-int

Syntax:

```
$(ipv4-to-int parameter)
```

Description: Converts the specified IPv4 address to its numeric representation. The numerical value of an IPv4 address is calculated by treating the IP address as a 4-byte hexadecimal value. For example, the 192.168.1.1 address equals to: 192=C0, 168=A8, 1=01, 1=01, or C0A80101, which is 3232235777 in decimal representation.

NOTE:

This template function is available only if the `convertfuncs` module has been loaded. By default, syslog-ng PE loads every available module.

By default, syslog-ng PE loads every available module. For details, see the section called "Loading modules"

Numeric operations

Syntax:

```
$(<operator> <first_operand> <second_operand>)
```

Description:

This template function performs simple numerical operations (like addition or multiplication) on integer numbers or macros containing integer numbers (for example, *${LEVEL_NUM}* or *${YEAR}*), and returns the result of the operation. The values used in the operation are not modified, that is, macros retain their original values. If one of the operands is not an integer, syslog-ng PE will not execute the template function, and return the **NaN** (Not-a-Number) value.

Available only in syslog-ng PE 4 F1 and later.

Operator	Operation:
+	Addition: returns the value of **<first_operand>+<second_operand>**
-	Subtraction: returns the value of **<first_operand>-<second_operand>**
*	Multiplication: returns the value of **<first_operand>*<second_operand>**
/	Division: returns the value of **<first_operand>/<second_operand>**
%	Modulus (remainder): returns the remainder of **<first_operand>/<second_operand>**

CAUTION:

The output of this template function is always an integer. If the return

value would be a floating-point number, the floating part is simply omitted, for example, 5.2 becomes 5, 5.8 becomes 5, -5.8 becomes -5.

It is also possible to nest numerical operations.

Example 14.12. Using numerical template functions

The following template function returns the facility of the log message multiplied by 8:

```
$(* ${FACILITY_NUM} 8)
```

Template functions can be nested into each other: the following template function returns the facility of the log message multiplied by 8, then adds this value to the severity of the message (the result is actually the priority of the message):

```
$(+ ${LEVEL_NUM} $(* ${FACILITY_NUM} 8))
```

strip

Syntax:

```
$(strip "<macro>")
```

Description: Deletes whitespaces from the beginning and the end of a macro. You can specify multiple macros separated with whitespace in a single template function, for example:

```
$(strip "${MESSAGE}" "${PROGRAM}")
```

Available in syslog-ng PE version 6.0.6 and later.

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: substituting (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.

Rewriting messages is often used in conjunction with message parsing Chapter 15, *Parsing and segmenting structured messages*.

Rewrite rules are similar to filters: they must be defined in the syslog-ng configuration file and used in the log statement. You can also define the rewrite rule inline in the log path.

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

Replacing message parts

To replace a part of the log message, you have to:

- define a string or regular expression to find the text to replace
- define a string to replace the original text (macros can be used as well)
- select the field of the message that the rewrite rule should process

Substitution rules can operate on any soft macros, for example MESSAGE, PROGRAM, or any user-defined macros created using parsers. Hard macros cannot be modified. For details on the hard and soft macros, see the section called “Hard vs. soft macros”). You can also 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>), 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 the section called “Regular expressions”.

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 14.13. Using substitution rules

The following example replaces the **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"));
 };
```

Example 14.14. Anonymizing IP addresses

The following example replaces every IPv4 address in the MESSAGE part with its SHA-1 hash:

```
rewrite  pseudonymize_ip_addresses_in_message  {
    subst  (
       "((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])[.]){3}([0-9]|[1-9]
[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]))",
       "$(sha1  $0)",
       value("MSG"),
       flags(global)
    );
 };
```

Setting message fields to specific values

To set a field of the message to a specific value, you have to:

- define the string to include in the message, and
- select the field where it should be included.

You can set the value of available macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (for details, see Chapter 15, *Parsing and segmenting structured messages* and Chapter 16, *Processing message content with a pattern database*). Hard macros cannot be modified. For details on the hard and soft macros, see the section called "Hard vs. soft macros". 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 14.15. 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 appends the "suffix" string to the MESSAGE field:

```
rewrite r_rewrite_set{set("$MESSAGE suffix", value("MESSAGE"));};
```

For details on rewriting SDATA fields, see the section called "Creating custom SDATA fields".

Creating custom SDATA fields

If you use RFC5424-formatted (IETF-syslog) messages, you can also create custom fields in the SDATA part of the message (For details on the SDATA message part, see the section called "The STRUCTURED-DATA message part"). According to RFC5424, the name of the field (its SD-ID) must not contain the `@` character for reserved SD-IDs. Custom SDATA fields must be in the following format: **`.SDATA.group-name@<private enterprise number>.field-name`**, for example, **`.SDATA.mySDATA-field-group@18372.4.mySDATA-field`**. (18372.4 is the private enterprise number of One Identity LLC, the developer of syslog-ng PE.)

Example 14.16. Rewriting custom SDATA fields

The following example sets the sequence ID field of the RFC5424-formatted (IETF-syslog) messages to a fixed value. This field is a predefined SDATA field with a reserved SD-ID, therefore its name does not contain the `@` character.

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

The next example creates a new SDATA field-group and field called **`custom`** and **`sourceip`**, respectively:

```
rewrite  r_rewrite_set  {
       set("${SOURCEIP}"  value(".SDATA.custom@18372.4.sourceip"));
 };
```

If you use the **`${.SDATA.custom@18372.4.sourceip}`** macro in a template or SQL table, its value will be that of the **`SOURCEIP`** macro (as seen on the machine where the SDATA field was created) for every message that was processed with this rewrite rule, and empty for every other message.

You can verify whether or not the format is correct by looking at the actual network traffic. The SDATA field-group will be called **`custom@18372.4`**, and **`sourceip`** will become a field within that group. If you decide to set up several fields, they will be listed in consecutive order within the field-group's SDATA block.

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}`**.

Setting multiple message fields to specific values

The *`groupset()`* rewrite rule allows you to modify the value of multiple message fields at once, for example, to change the value of sensitive fields extracted using patterndb, or received in a JSON format.

- The first parameter is the new value of the modified fields. This can be a simple string, a macro, or a template (which can include template functions as well).
- The second parameter (*values()*) specifies the fields to modify. You can explicitly list the macros or fields (a space-separated list with the values enclosed in double-quotes), or use wildcards and glob expressions to select multiple fields.
- Note that *groupset()* does not create new fields, it only modifies existing fields.

  You can refer to the old value of the field using the *$_* macro. This is resolved to the value of the current field, and is available only in *groupset()* rules.

Declaration:

```
rewrite <name_of_the_rule> {
      groupset("<new-value-of-the-fields>", values("<field-name-or-glob>"
["<another-field-name-or-glob>"]));
 };
```

Example 14.17. Using groupset rewrite rules

The following examples show how to change the values of multiple fields at the same time.

- Change the value of the *HOST* field to *myhost*.

  ```
  groupset ("myhost" values("HOST"))
  ```
- Change the value of the *HOST* and *FULLHOST* fields to *myhost*.

  ```
  groupset ("myhost" values("HOST" "FULLHOST"))
  ```
- Change the value of the *HOSTFULLHOST* and fields to lowercase.

  ```
  groupset ("$(lowercase "$_")" values("HOST" "FULLHOST"))
  ```
- Change the value of each field and macro that begins with *.USER* to *nobody*.

  ```
  groupset ("nobody" values(".USER.*"))
  ```
- Change the value of each field and macro that begins with *.USER* to its SHA-1 hash (truncated to 6 characters).

  ```
  groupset ("$(sha1 --length 6 $_)" values(".USER.*"))
  ```

Conditional rewrites

Starting with 4 F1, it is possible to apply a rewrite rule to a message only if certain conditions are met. The *condition()* option effectively embeds a filter expression into the rewrite rule: the message is modified only if the message passes the filter. If the condition is not met, the message is passed to the next element of the log path (that is, the element following the rewrite rule in the log statement, for example, the destination). Any filter expression normally used in filters can be used as a rewrite condition. Existing filter statements can be referenced using the *filter()* function within the condition. For details on filters, see the section called “Filters”.

TIP:

Using conditions in rewrite rules can simplify your syslog-ng PE configuration file, as you do not need to create separate log paths to modify certain messages.

Procedure 14.1. How conditional rewriting works

Purpose:

The following procedure summarizes how conditional rewrite rules (rewrite rules that have the *condition()* parameter set) work. The following configuration snippet is used to illustrate the procedure:

```
rewrite r_rewrite_set{set("myhost", value("HOST") condition(program
("myapplication")));};
log {
    source(s1);
    rewrite(r_rewrite_set);
    destination(d1);};
```

Steps:

1. The log path receives a message from the source (**s1**).
2. The rewrite rule (**r_rewrite_set**) evaluates the condition. If the message matches the condition (the PROGRAM field of the message is "myapplication"), syslog-ng PE rewrites the log message (sets the value of the HOST field to "myhost"), otherwise it is not modified.
3. The next element of the log path processes the message (**d1**).

Example 14.18. Using conditional rewriting

The following example sets the HOST field of the message to **myhost** only if the message was sent by the **myapplication** program.

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

The following example is identical to the previous one, except that the condition references an existing filter template.

```
filter f_rewritefilter {program("myapplication");};
rewrite r_rewrite_set{set("myhost", value("HOST") condition(filter(f_
rewritefilter)));};
```

Regular expressions

Filters and substitution rewrite rules can use regular expressions. In regular expressions, the characters `()[].*?+^$|\` are used as special symbols. Depending on how you want to use these characters and which quotation mark you use, these characters must be used differently, as summarized below.

- Strings between single quotes (`'string'`) are treated literally and are not interpreted at all, you do not have to escape special characters. For example the output of `'\x41'` is `\x41` (characters as follows: backslash, `x`(letter), `4`(number), `1` (number)). 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"`), the string is interpreted and you have to escape special characters, that is, to precede them with a backslash (\) character if they are meant literally. For example the output of the `"\x41"` is simply the letter `a`. Therefore special characters like \(backslash) or "(quotation mark) must be escaped (\\ and \"). The following expressions are interpreted: `\a`, `\n`, `\r`, `\t`, `\v`. 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, you do not need to escape the backslash, for example `match("\\.")` is equivalent to `match('\.')`.

- Enclosing alphanumeric 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.

  NOTE:

  Only strings containing alphanumerical characters can be used without quotes or double quotes. If the string contains whitespace or any special characters (`()[].*?+^$|\` or `;:#`), you must use quotes or double quotes.

  When using the `;:#` characters, you must use quotes or double quotes, but escaping them is not required.

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)); };
```

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.

Types and options of regular expressions

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 PE 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 `$0, ... $255` variables. The `$0` stores the entire match, `$1` is the first group of the match (parentheses), and so on. Matches from the last filter expression can be referenced in regular expressions.

utf8: Use UTF-8 matching.

Example 14.19. Using Posix regular expressions

```
filter f_message { message("keyword" flags("utf8" "ignore-case") ); };
```

pcre

Description: Use Perl Compatible Regular Expressions (PCRE). Starting with syslog-ng PE 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.

store-matches: Store the matches of the regular expression into the `$0, ... $255` variables. The `$0` stores the entire match, `$1` is the first group of the match (parentheses), and so on. 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 14.20. Using PCRE regular expressions

```
rewrite r_rewrite_subst
        {subst("a*", "?", value("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: Match the strings against a pattern containing '*' and '?' wildcards, without regular expression and character range support. The advantage of glob patterns to regular expressions is that globs can be processed much faster.

`*`

matches an arbitrary string, including an empty string

`?`

matches an arbitrary character

NOTE:

- The wildcards can match the `/` character.
- You cannot use the `*` and `?` literally in the pattern.

Optimizing regular expressions

The *host()*, *match()*, and *program()* filter functions and some other syslog-ng objects 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 14.21. 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"); };
```

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.

syslog-ng PE provides the following possibilities to parse the messages, or parts of the messages:

- To segment a message into columns using a CSV-parser, see the section called "Parsing messages with comma-separated and similar values".
- To segment a message consisting of whitespace or comma-separated **`key=value`** pairs (for example, Postfix log messages), see the section called "Parsing **`key=value`** pairs".
- To parse JSON-formatted messages, see the section called "The JSON parser".
- To identify and parse the messages using a pattern database, see Chapter 16, *Processing message content with a pattern database*.

Parsing messages with comma-separated and similar values

The syslog-ng PE application can separate parts of log messages (that is, the contents of the ${MSG} macro) at delimiter characters or strings to named fields (columns). One way to achieve this is to use a csv (comma-separated-values) parser (for other methods and possibilities, see the other sections of Chapter 15, *Parsing and segmenting structured messages*. The parsed fields act as user-defined macros that can be referenced in message templates, file- and tablenames, and so on.

Parsers are similar to filters: they must be defined in the syslog-ng PE configuration file and used in the log statement. You can also define the parser inline in the log path.

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

To create a *`csv-parser()`*, you have to define the columns of the message, the separator characters or strings (also called delimiters, for example, semicolon or tabulator), and optionally the characters that are used to escape the delimiter characters (*`quote-pairs()`*).

Declaration:

```
parser <parser_name> {
      csv-parser(
      columns(column1, column2, ...)
      delimiters(chars("<delimiter_characters>"), strings("<delimiter_string1>"))
      );
};
```

Column names work like macros.

Names starting with a dot (for example, **.example**) are reserved for use by syslog-ng PE. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see the section called "Hard vs. soft macros" for details).

Example 15.1. 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 15.2. 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 15.3. Segmenting a part of a message

Multiple parsers can be used to split a part of an already parsed message into further segments. 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);  parser(p_apache);  parser(p_apache_timestamp);
destination(d_file);
 };
```

Further examples:

- For an example on using the *greedy* option, see Example 15.4, "Adding the end of the message to the last column".

Options of CSV parsers

The syslog-ng PE application can separate parts of log messages (that is, the contents of the ${MSG} macro) at delimiter characters or strings to named fields (columns). One way to achieve this is to use a csv (comma-separated-values) parser (for other methods and possibilities, see the other sections of Chapter 15, *Parsing and segmenting structured messages*. The parsed fields act as user-defined macros that can be referenced in message templates, file- and tablenames, and so on.

Parsers are similar to filters: they must be defined in the syslog-ng PE configuration file and used in the log statement. You can also define the parser inline in the log path.

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

To create a *csv-parser()*, you have to define the columns of the message, the separator characters or strings (also called delimiters, for example, semicolon or tabulator), and optionally the characters that are used to escape the delimiter characters (*quote-pairs()*).

Declaration:

```
parser <parser_name> {
      csv-parser(
      columns(column1, column2, ...)
      delimiters(chars("<delimiter_characters>"), strings("<delimiter_string1>"))
      );
};
```

Column names work like macros.

Names starting with a dot (for example, **.example**) are reserved for use by syslog-ng PE. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see the section called "Hard vs. soft macros" for details).

columns()

Synopsis: columns("PARSER.COLUMN1", "PARSER.COLUMN2", ...)

Description: Specifies the name of the columns to separate messages to. These names will be automatically available as macros. The values of these macros do not include the delimiters.

delimiters()

Synopsis: delimiters(chars("<delimiter_characters>")) *or* *delimiters("<delimiter_characters>")*

delimiters(strings("<delimiter_string1>", "<delimiter_string2>", ...)")

delimiters(chars("<delimiter_characters>"), strings("<delimiter_string1>"))

Description: The delimiter is the character or string that separates the columns in the message. If you specify multiple characters using the **delimiters(chars("<delimiter_characters>"))** option, every character will be treated as a delimiter. To separate the columns at the tabulator (tab character), specify **\t**. For example, to separate the text at every hyphen (-) and colon (:) character, use **delimiters(chars("-:"))**, Note that the delimiters will not be included in the column values.

If you have to use a string as a delimiter, list your string delimiters in the **delimiters(strings("<delimiter_string1>", "<delimiter_string2>", ...)")** format.

If you use more than one delimiter, note the following points:

- syslog-ng PE will split the message at the nearest possible delimiter. The order of the delimiters in the configuration file does not matter.
- You can use both string delimiters and character delimiters in a parser.
- The string delimiters can include characters that are also used as character delimiters.
- If a string delimiter and a character delimiter both match at the same position of the message, syslog-ng PE uses the string delimiter.

flags()

Synopsis: drop-invalid, escape-none, escape-backslash, escape-double-char, greedy, strip-whitespace

Description: Specifies various options for parsing the message. The following flags are available:

- *drop-invalid*: When the *drop-invalid* option is set, the parser does not process messages that do not match the parser. For example, a message does not match the parser if it has less columns than specified in the parser, or it has more columns but the *greedy* flag is not enabled. Using the *drop-invalid* option practically turns the

parser into a special filter, that matches messages that have the predefined number of columns (using the specified delimiters).

> TIP:
>
> Messages dropped as invalid can be processed by a *fallback* log path. For details on the *fallback* option, see the section called "Log path flags".

- *escape-backslash*: The parsed message uses the backslash (\) character to escape quote characters.
- *escape-double-char*: The parsed message repeats the quote character when the quote character is used literally. For example, to escape a comma (,), the message contains two commas (,,).
- *escape-none*: The parsed message does not use any escaping for using the quote character literally.
- *greedy*: 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.

Example 15.4. 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 (that is, **some more information**) is not parsed, and possibly lost if you use only the variables to reconstruct 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));
```

strip-whitespace: The `strip-whitespace` flag removes leading and trailing whitespaces from all columns.

quote-pairs()

Synopsis: quote-pairs('<quote_pairs>')

Description: List quote-pairs between single quotes. Delimiter characters or strings 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. For an example of using `quote-pairs()` to parse Apache log files, see Example 15.2, "Parsing Apache log files".

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 (`${MESSAGE}`). For examples, see Example 15.1, "Segmenting hostnames separated with a dash" and Example 15.3, "Segmenting a part of a message".

Parsing `key=value` pairs

The syslog-ng PE application can separate a message consisting of whitespace or comma-separated **`key=value`** pairs (for example, Postfix log messages) into name-value pairs. You can also specify other separator character instead of the equal sign, for example, colon (`:`) to parse MySQL log messages. For details on using value-pairs in syslog-ng PE see the section called "Structuring macros, metadata, and other value-pairs".

You can refer to the separated parts of the message using the key of the value as a macro. For example, if the message contains **`KEY1=value1,KEY2=value2`**, you can refer to the values as **`${KEY1}`** and **`${KEY2}`**.

NOTE:

If a log message contains the same key multiple times (for example, **`key1=value1, key2=value2, key1=value3, key3=value4, key1=value5`**), then syslog-ng PE stores only the last (rightmost) value for the key. Using the previous example, syslog-ng PE will store the following pairs: **`key1=value5, key2=value2, key3=value4.`**.

CAUTION:

If the names of keys in the message is the same as the names of syslog-ng PE soft macros, the value from the parsed message will overwrite the value of the macro. For example, the `PROGRAM=value1, MESSAGE=value2` content will overwrite the `${PROGRAM}` and `${MESSAGE}` macros. To avoid overwriting such macros, use the *`prefix()`* option.

Hard macros cannot be modified, so they will not be overwritten. For details on the macro types, see the section called "Hard vs. soft macros".

The parser discards message sections that are not `key=value` pairs, even if they appear between `key=value` pairs that can be parsed.

The names of the keys can contain only the following characters: numbers (0-9), letters (a-z,A-Z), underscore (_), dot (.), hyphen (-). Other special characters are not permitted.

To parse **`key=value`** pairs, define a parser that has the *`kv-parser()`* option. Defining the prefix is optional. By default, the parser will process the **`${MESSAGE}`** part of the log message. You can also define the parser inline in the log path.

Declaration:

```
parser  parser_name  {
       kv-parser(
             prefix()
       );
};
```

Example 15.5. Using a `key=value` parser

In the following example, the source is a log message consisting of comma-separated **`key=value`** pairs, for example, a Postfix log message:

```
Jun 20 12:05:12 mail.example.com <info> postfix/qmgr[35789]: EC2AC1947DA:
from=<me@example.com>, size=807, nrcpt=1 (queue active)
```

The kv-parser inserts the "**`.kv.`**" prefix before all extracted name-value pairs. The destination is a file, that uses the *`format-json`* template function. Every name-value pair that begins with a dot (".") character will be written to the file (**`dot-nv-pairs`**). The log line connects the source, the destination and the parser.

```
source  s_kv  {
       network(port(21514));
};

destination  d_json  {
       file("/tmp/test.json"
             template("$(format-json  --scope  dot-nv-pairs)\n"));
};

parser  p_kv  {
       kv-parser  (prefix(".kv."));
};

log  {
       source(s_kv);
       parser(p_kv);
       destination(d_json);
};
```

You can also define the parser inline in the log path.

```
source s_kv {
    network(port(21514));
};

destination d_json {
    file("/tmp/test.json"
        template("$(format-json --scope dot-nv-pairs)\n"));
};

log {
    source(s_kv);
```

```
    parser {
        kv-parser (prefix(".kv."));
    };
    destination(d_json);
};
```

You can set the separator character between the key and the value to parse for example **key:value** pairs, like MySQL logs:

```
Mar   7 12:39:25 myhost MysqlClient[20824]: SYSTEM_USER:'oscar', MYSQL_
USER:'my_oscar', CONNECTION_ID:23, DB_SERVER:'127.0.0.1', DB:'--',
QUERY:'USE test;'
```

```
parser p_mysql { kv-parser(value-separator(":") prefix(".mysql.")); };
```

Options of `key=value` parsers

The *kv-parser* has the following options.

prefix()

Synopsis: prefix()

Description: Insert a prefix before the name part of the parsed name-value pairs to help further processing. For example:

- To insert the **my-parsed-data.** prefix, use the **prefix(my-parsed-data.)** option.
- To refer to a particular data that has a prefix, use the prefix in the name of the macro, for example, **${my-parsed-data.name}** .
- If you forward the parsed messages using the IETF-syslog protocol, you can insert all the parsed data into the SDATA part of the message using the **prefix(.SDATA.my-parsed-data.)** option.

Names starting with a dot (for example, **.example**) are reserved for use by syslog-ng PE. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see the section called “Hard vs. soft macros” for details). To avoid such problems, use a prefix when naming the parsed values, for example, **prefix(my-parsed-data.)**

For example, to insert the **postfix** prefix when parsing Postfix log messages, use the **prefix(.postfix.)** option.

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, the parser processes the entire message (`${MESSAGE}`).

value-separator()

Synopsis: value-separator()

Description: Specifies the character that separates the keys from the values. Default value: `=`

For example, to parse `key:value` pairs, use `kv-parser(value-separator(":"));`

The JSON parser

JavaScript Object Notation (JSON) is a text-based open standard designed for human-readable data interchange. It is used primarily to transmit data between a server and web application, serving as an alternative to XML. It is described in RFC 4627. The syslog-ng PE application can separate parts of incoming JSON-encoded log messages to name-value pairs. For details on using value-pairs in syslog-ng PE see the section called "Structuring macros, metadata, and other value-pairs".

You can refer to the separated parts of the JSON message using the key of the JSON object as a macro. For example, if the JSON contains **`{"KEY1":"value1","KEY2":"value2"}`**, you can refer to the values as **`${KEY1}`** and **`${KEY2}`**. If the JSON content is structured, syslog-ng PE converts it to dot-notation-format. For example, to access the value of the following structure **`{"KEY1": {"KEY2": "VALUE"}}`**, use the **`${KEY1.KEY2}`** macro.

CAUTION:

If the names of keys in the JSON content are the same as the names of syslog-ng PE soft macros, the value from the JSON content will overwrite the value of the macro. For example, the `{"PROGRAM":"value1","MESSAGE":"value2"}` JSON content will overwrite the `${PROGRAM}` and `${MESSAGE}` macros. To avoid overwriting such macros, use the *`prefix()`* option.

Hard macros cannot be modified, so they will not be overwritten. For details on the macro types, see the section called "Hard vs. soft macros".

NOTE:

The JSON parser currently supports only integer, double and string values when interpreting JSON structures. As syslog-ng does not handle different data types internally, the JSON parser converts all JSON data to string values. In case of boolean types, the value is converted to 'TRUE' or 'FALSE' as their string representation.

The JSON parser discards messages if it cannot parse them as JSON messages, so it acts as a JSON-filter as well.

To create a JSON parser, define a parser that has the *`json-parser()`* option. Defining the prefix and the marker are optional. By default, the parser will process the **`${MESSAGE}`** part of the log message. To process other parts of a log message with the JSON parser, use the *`template()`* option. You can also define the parser inline in the log path.

Declaration:

```
parser  parser_name  {
       json-parser(
       marker()
       prefix()
       );
 };
```

Example 15.6. Using a JSON parser

In the following example, the source is a JSON encoded log message. The syslog parser is disabled, so that syslog-ng PE does not parse the message: **flags(no-parse)**. The json-parser inserts "**.json.**" prefix before all extracted name-value pairs. The destination is a file, that uses the *format-json* template function. Every name-value pair that begins with a dot (".") character will be written to the file (**dot-nv-pairs**). The log line connects the source, the destination and the parser.

```
source s_json {
      network(port(21514) flags(no-parse));
};

destination d_json {
      file("/tmp/test.json"
            template("$(format-json --scope dot-nv-pairs)\n"));
};

parser p_json {
      json-parser (prefix(".json."));
};

log {
      source(s_json);
      parser(p_json);
      destination(d_json);
};
```

You can also define the parser inline in the log path.

```
source s_json {
      network(port(21514) flags(no-parse));
};

destination d_json {
      file("/tmp/test.json"
            template("$(format-json --scope dot-nv-pairs)\n"));
};

log {
      source(s_json);
      parser p_json {
            json-parser (prefix(".json."));
      };
      destination(d_json);
};
```

Options of JSON parsers

The JSON parser has the following options.

extract-prefix()

Synopsis: extract-prefix()

Description: Extract only the specified subtree from the JSON message. Use the dot-notation to specify the subtree. The rest of the message will be ignored. For example, assuming that the incoming object is named `msg`, the `json-parser(extract-prefix("foo.bar[5]"));` syslog-ng PE parser is equivalent to the `msg.foo.bar[5]` javascript code. Note that the resulting expression must be a JSON object, so that syslog-ng PE can extract its members into name-value pairs.

This feature also works when the top-level object is an array, because you can use an array index at the first indirection level, for example: `json-parser(extract-prefix("[5]"))`, which is equivalent to `msg[5]`.

marker()

Synopsis: marker()

Description: Use a marker in case of mixed log messages, to identify JSON encoded messages for the parser.

Some logging implementations require a marker to be set before the JSON payload. The JSON parser is able to find these markers and parse the message only if it is present.

Example 15.7. Using the marker option in JSON parser

This json parser parses log messages which use the "@cee:" marker in front of the json payload. It inserts "`.cee.`" in front of the name of name-value pairs, so later on it is easier to find name-value pairs that were parsed using this parser. (For details on selecting name-value pairs, see the section called “value-pairs()”.)

```
parser {
            json-parser(
                  marker("@cee:")
                  prefix(".cee.")
            );
      };
```

prefix()

Synopsis: prefix()

Description: Insert a prefix before the name part of the name-value pairs to help further processing. For example, if you forward the parsed messages using the IETF-syslog protocol, you can insert all the parsed data into the SDATA part of the message using the **`prefix(.SDATA.json.)`**.

Names starting with a dot (for example, **`.example`**) are reserved for use by syslog-ng PE. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see the section called "Hard vs. soft macros" for details). To avoid such problems, use a prefix when naming the parsed values, for example, **`prefix(my-parsed-data.)`**

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 (**`${MESSAGE}`**).

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, and so on 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, and so on). Other pattern parsers match on various strings and IP addresses. For the details of available pattern parsers, see the section called “Using pattern parsers”.

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))?`**

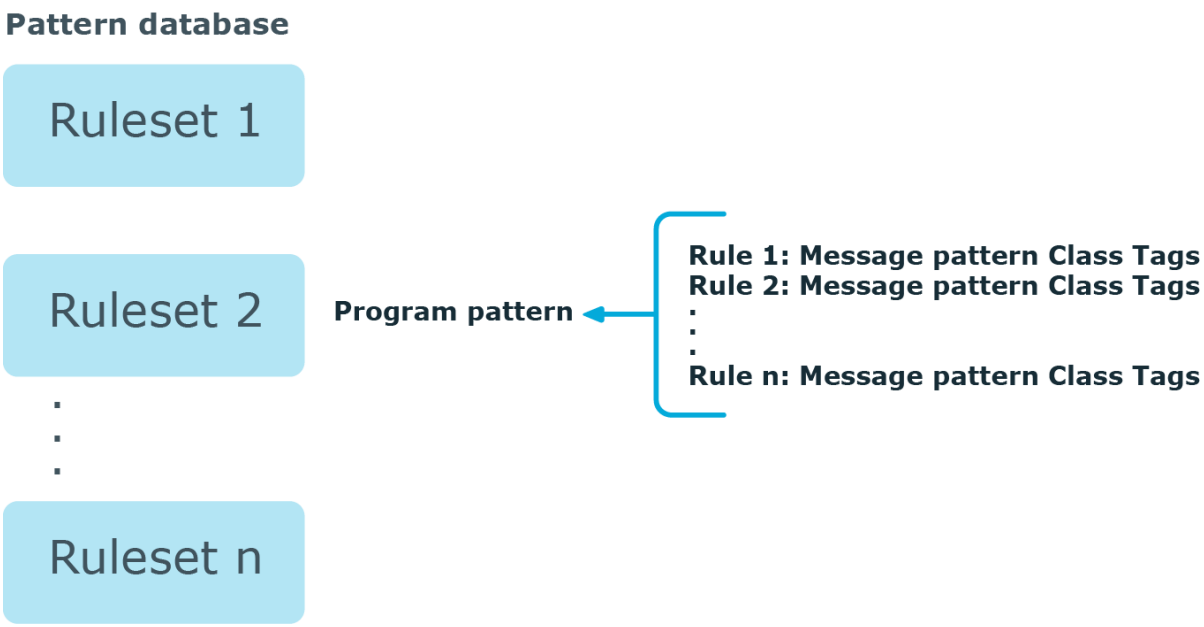
A syslog-ng database pattern for this message: **`Accepted @QSTRING:auth_method: @for@QSTRING:username: @from @QSTRING:client_addr: @port @NUMBER:port:@ ssh2`**

For details on using pattern databases to classify log messages, see the section called “Using pattern databases”.

The structure of the pattern database

The pattern database is organized as follows:

Figure 16.1. 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 (for example, 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 (for example, Security, Violation, and so on).
- 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

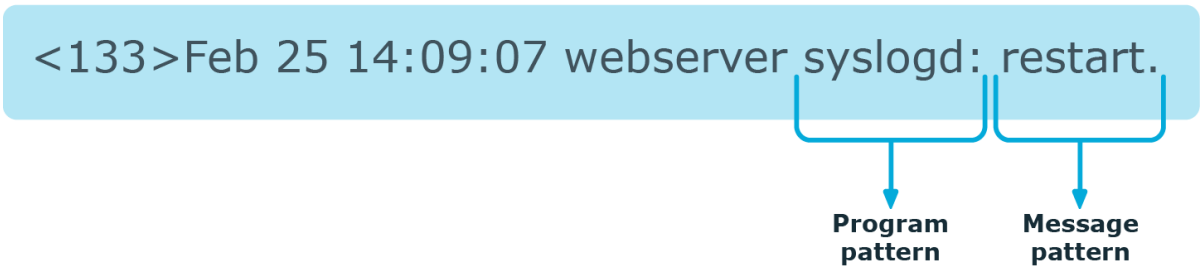
Pattern are used to classify the message.

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.

How pattern matching works

Figure 16.2. Applying patterns

A sample log message:



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, that is, 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. If the literal does not match the incoming string (for example, **Apple**), syslog-ng attempts to match the pattern with the parser. However, if there are two or more parsers on the same level, only the first one will be applied, even if it does not perfectly match the message.

If there are two parsers at the same level (for example, **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.

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 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. For sample pattern databases, see the section called “Downloading sample pattern databases”.

Using pattern databases

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 16.1. 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 16.2. 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)
       );
 };
```

For details on how to create your own pattern databases see the section called “The syslog-ng pattern database format”.

Using parser results in filters and templates

The results of message classification and parsing can be used in custom filters and templates, for example, in file and database templates. The following built-in macros 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).
- The *.classifier.rule_id* macro contains the identifier of the message pattern that matched the message.

Example 16.3. 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")
                    );
                    };
```

Pattern database rules can assign tags to messages. These tags can be used to select tagged messages using the *tags()* filter function.

NOTE:

The syslog-ng PE application automatically adds the class of the message as a tag using the *.classifier.<message-class>* format. For example, messages classified as "system" receive the *.classifier.system* tag. Use the *tags()* filter function to select messages of a specific class.

```
filter f_tag_filter {tags(".classifier.system");};
```

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 16.4. 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, and so on). 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 log path 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**.

Downloading sample pattern databases

To simplify the building of pattern databases, Balabit has released (and will continue to release) sample databases. You can download sample pattern databases from the PatternDB GitHub page.

Note that these pattern databases are only samples and experimental databases. They 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-Share Alike 3.0 (CC by-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 your needs.
- If you redistribute the pattern databases, you must distribute your modifications under the same license.
- If you redistribute the pattern databases, you must make it obvious that the source of the original syslog-ng pattern databases is the PatternDb GitHub page.

For legal details, the full text of the license is available here.

If you create patterns that are not available in the GitHub repository, consider sharing them with us and the syslog-ng community, and send them to the syslog-ng mailing list, or to the following e-mail address:<patterndb@balabit.com>

Correlating log messages

The syslog-ng PE application is able to correlate log messages identified using pattern databases.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix e-mail server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message.

Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages, syslog-ng PE uses the pattern database to add messages into message-groups called contexts. A context consists of a series of log messages that are related to each other in some way, for example, the log messages of an SSH session can belong to the same context. As new messages come in, they may be added to a context. Also, when an incoming message is identified it can trigger actions to be performed, for example, generate a new message that contains all the important information that was stored previously in the context. (For details on triggering actions and generating messages, see the section called "Triggering actions for identified messages".)

There are two attributes for pattern database rules that determine if a message matching the rule is added to a context: *context-scope* and *context-id*. The *context-scope* attribute acts as an early filter, selecting messages sent by the same process (${HOST}${PROGRAM}${PID} is identical), application (${HOST}${PROGRAM} is identical), or host, while the *context-id* actually adds the message to the context specified in the id. The *context-id* can be a simple string, or can contain macros or values extracted from the log messages for further filtering.

> NOTE:
>
> Message contexts are persistent and are not lost when syslog-ng PE is reloaded (SIGHUP), but are lost when syslog-ng PE is restarted.

Another parameter of a rule is the *context-timeout* attribute, which determines how long a context is stored, that is, how long syslog-ng PE waits for related messages to arrive. Note the following points about timeout values:

- When a new message is added to a context, syslog-ng PE will restart the timeout using the *context-timeout* set for the new message.
- When calculating if the timeout has already expired or not, syslog-ng PE uses the timestamps of the incoming messages, not system time elapsed between receiving the two messages (unless the messages do not include a timestamp, or the **keep-timestamp(no)** option is set). That way syslog-ng PE can be used to process and correlate already existing log messages offline. However, the timestamps of the messages must be in chronological order (that is, a new message cannot be older

than the one already processed), and if a message is newer than the current system time (that is, it seems to be coming from the future), syslog-ng PE will replace its timestamp with the current system time.

Example 16.5. How syslog-ng PE calculates *context-timeout*

Consider the following two messages:

```
<38>1990-01-01T14:45:25 customhostname program6[1234]: program6
testmessage
<38>1990-01-01T14:46:25 customhostname program6[1234]: program6
testmessage
```

If the *context-timeout* is 10 seconds and syslog-ng PE receives the messages within 1 sec, the timeout event will occour immediately, because the difference of the two timestamp (60 sec) is larger than the timeout value (10 sec).

- Avoid using unnecessarily long timeout values on high-traffic systems, as storing the contexts for many messages can require considerable memory. For example, if two related messages usually arrive within seconds, it is not needed to set the timeout to several hours.

Example 16.6. Using message correlation

```
<rule xml:id="..." context-id="ssh-session" context-timeout="86400"
context-scope="process">      <patterns>            <pattern>Accepted
@ESTRING:usracct.authmethod: @for @ESTRING:usracct.username: @from
@ESTRING:usracct.device: @port @ESTRING::
@@ANYSTRING:usracct.service@</pattern>      </patterns>...
 </rule>
```

For details on configuring message correlation, see the description of the context-id, context-timeout, and context-scope attributes of pattern database rules.

Referencing earlier messages of the context

When using the **<value>** element in pattern database rules together with message correlation, you can also refer to fields and values of earlier messages of the context by adding the **@<distance-of-referenced-message-from-the-current>** suffix to the macro. For example, if there are three log messages in a context, and you are creating a

generated message for the third log message, the **${HOST}@1** expression refers to the host field of the current (third) message in the context, the **${HOST}@2** expression refers to the host field of the previous (second) message in the context, **${PID}@3** to the PID of the first message, and so on. For example, the following message can be created from SSH login/logout messages (for details on generating new messages, see the section called "Triggering actions for identified messages"): **An SSH session for ${SSH_USERNAME}@1 from ${SSH_CLIENT_ADDRESS}@2 closed. Session lasted from ${DATE}@2 to ${DATE}.**

CAUTION:

When referencing an earlier message of the context, always enclose the field name between braces, for example, `${PID}@3`. The reference will not work if you omit the braces.

NOTE:

To use a literal `@` character in a template, use `@@`.

Example 16.7. Referencing values from an earlier message

The following action can be used to log the length of an SSH session (the time difference between a login and a logout message in the context):

```
<actions>    <action>        <message>            <values>
<value name="MESSAGE">An SSH session for ${SSH_USERNAME}@1 from ${SSH_CLIENT_
ADDRESS}@2 closed. Session lasted from ${DATE}@2 ${DATE} </value>
</values>        </message>    </action></actions>
```

Triggering actions for identified messages

The syslog-ng PE application is able to generate (trigger) messages automatically if certain events occur, for example, a specific log message is received, or the correlation timeout of a message expires. Basically, you can define messages for every pattern database rule that are emitted when a message matching the rule is received. Triggering messages is often used together with message correlation, but can also be used separately.

The generated message is injected into the same place where the *db-parser()* statement is referenced in the log path. To post the generated message into the *internal()* source instead, use the *inject-mode()* option in the definition of the parser.

Example 16.8. Sending triggered messages to the *internal()* source

To send the generated messages to the *internal* source, use the **inject-mode (internal)** option:

```
parser  p_db  {db-parser(
       file("mypatterndbfile.xml")
       inject-mode(internal)
 );};
```

To inject the generated messages where the pattern database is referenced, use the **inject-mode(pass-through)** option:

```
parser  p_db  {db-parser(
       file("mypatterndbfile.xml")
       inject-mode(pass-through)
 );};
```

The generated message must be configured in the pattern database rule. It is possible to create an entire message, use macros and values extracted from the original message with pattern database, and so on.

Example 16.9. Generating messages for pattern database matches

When inserted in a pattern database rule, the following example generates a message when a message matching the rule is received.

```
<actions>    <action>        <message>            <values>
<value name="MESSAGE">A log message from ${HOST} matched rule number
$.classifier.rule_id</value>            </values>        </message>
</action></actions>
```

To inherit the properties and values of the triggering message, set the *inherit-properties* attribute of the *<message>* element to TRUE. That way the triggering log message is cloned, including name-value pairs and tags. If you set any values for the message in the *<action>* element, they will override the values of the original message.

Example 16.10. Generating messages with inherited values

The following action generates a message that is identical to the original message, but its $PROGRAM field is set to *overriding-original-program-name*

```
<actions>    <action>        <message inherit-properties='TRUE'>
<values>                <value name="PROGRAM">overriding-original-program-
name</value>            </values>        </message>    </action></actions>
```

For details on configuring actions, see the description of the pattern database format.

Conditional actions

To limit when a message is triggered, use the *condition* attribute and specify a filter expression: the action will be executed only if the condition is met. For example, the following action is executed only if the message was sent by the host called **myhost**.

```
<action condition="'${HOST}' == 'example'">
```

You can use the same operators in the condition that can be used in filters. For details, see the section called “Comparing macro values in filters”.

The following action can be used to log the length of an SSH session (the time difference between a login and a logout message in the context):

```
<actions>    <action>        <message>            <values>                <value
name="MESSAGE">An SSH session for ${SSH_USERNAME}@1 from ${SSH_CLIENT_ADDRESS}@2
closed. Session lasted from ${DATE}@2 ${DATE} </value>            </values>
</message>    </action></actions>
```

Example 16.11. Actions based on the number of messages

The following example triggers different actions based on the number of messages in the context. This way you can check if the context contains enough messages for the event to be complete, and execute a different action if it does not.

```
<actions>    <action condition='"$(context-length)" >= "4"'>        <message>
          <values>                <value name="PROGRAM">event</value>
      <value name="MESSAGE">Event complete</value>            </values>
</message>    </action>    <action condition='"$(context-length)" < "4"'>
  <message>            <values>                <value
name="PROGRAM">error</value>            <value name="MESSAGE">Error
detected</value>            </values>        </message>    </action></actions>
```

External actions

To perform an external action when a message is triggered, for example, to send the message in an e-mail, you have to route the generated messages to an external application using the *program()* destination.

Example 16.12. Sending triggered messages to external applications

The following sample configuration selects the triggered messages and sends them to an external script.

1. Set a field in the triggered message that is easy to identify and filter. For example:

   ```
   <values>    <value name="MESSAGE">A log message from ${HOST} matched
   rule number $.classifier.rule_id</value>    <value
   name="TRIGGER">yes</value></values>
   ```

2. Create a destination that will process the triggered messages.

   ```
   destination d_triggers { program("/bin/myscript"; ); };
   ```

3. Create a filter that selects the triggered messages from the internal source.

   ```
   filter f_triggers {match("yes" value ("TRIGGER") type(string));};
   ```

4. Create a log path that selects the triggered messages from the internal source

and sends them to the script:

```
log { source(s_local); filter(f_triggers); destination(d_triggers); };
```

5. Create a script that will actually process the generated messages, for example:

```
#!/usr/bin/perl
 while (<>) {
             # body of the script to send emails, snmp traps, and
so on
 }
```

Actions and message correlation

Certain features of generating messages can be used only if message correlation is used as well. For details on correlating messages, see the section called "Correlating log messages".

- The syslog-ng PE application automatically fills the fields for the generated message based on the scope of the context, for example, the HOST and PROGRAM fields if the *context-scope* is **program**.

  When used together with message correlation, you can also refer to fields and values of earlier messages of the context by adding the **@<distance-of-referenced-message-from-the-current>** suffix to the macro. For details, see the section called "Referencing earlier messages of the context".

  Example 16.13. Referencing values from an earlier message

  The following action can be used to log the length of an SSH session (the time difference between a login and a logout message in the context):

  ```
  <actions>    <action>        <message>            <values>
   <value name="MESSAGE">An SSH session for ${SSH_USERNAME}@1 from ${SSH_
  CLIENT_ADDRESS}@2 closed. Session lasted from ${DATE}@2 ${DATE} </value>
            </values>        </message>    </action></actions>
  ```

- You can use the name-value pairs of other messages of the context. If you set the *inherit-properties* attribute of the generated message to *context*, syslog-ng PE

collects every name-value pair from each message stored in the context, and includes them in the generated message. This means that you can refer to a name-value pair without having to know which message of the context included it. If a name-value pair appears in multiple messages of the context, the value in the latest message will be used. To refer to an earlier value, use the **@<distance-of-referenced-message-from-the-current>** suffix format.

```
<action>    <message inherit-properties='context'>
```

Example 16.14. Using the *inherit-properties* option

For example, if *inherit-properties* is set to **context**, and you have a rule that collects SSH login and logout messages to the same context, you can use the following value to generate a message collecting the most important information form both messages, including the beginning and end date.

```
<value name="MESSAGE">An SSH session for ${SSH_USERNAME} from ${SSH_
CLIENT_ADDRESS} closed. Session lasted from ${DATE}@2 to $DATE pid:
$PID.</value>
```

The following is a detailed rule for this purpose.

```
<patterndb version='4' pub_date='2015-04-13'>    <ruleset name='sshd'
id='12345678'>        <pattern>sshd</pattern>            <rules>
       <!-- The pattern database rule for the first log message -->
          <rule provider='me' id='12347598' class='system'
                   context-id="ssh-login-logout" context-
timeout="86400"
                   context-scope="process">               <!-- Note
the context-id that groups together the
                relevant messages, and the context-timeout value that
                determines how long a new message can be added to the
                context  -->                    <patterns>
         <pattern>Accepted @ESTRING:SSH.AUTH_METHOD: @for @ESTRING:SSH_
USERNAME: @from @ESTRING:SSH_CLIENT_ADDRESS: @port @ESTRING::
@@ANYSTRING:SSH_SERVICE@</pattern>                        <!-- This is
the actual pattern used to identify
                        the log message. The segments between the @
                        characters are parsers that recognize the
variable
                        parts of the message - they can also be used as
                        macros.  -->                    </patterns>
          </rule>               <!-- The pattern database rule for the
```

```
fourth log message -->                <rule provider='me' id='12347599'
class='system' context-id="ssh-login-logout" context-scope="process">
               <patterns>                         <pattern>pam_unix
(sshd:session): session closed for user @ANYSTRING:SSH_
USERNAME@</pattern>                    </patterns>
<actions>                         <action>
<message inherit-properties='context'>
<values>                                    <value name="MESSAGE">An SSH
session for ${SSH_USERNAME} from ${SSH_CLIENT_ADDRESS} closed. Session
lasted from ${DATE}@2 to $DATE pid: $PID.</value>
            <value name="TRIGGER">yes</value>
        <!-- This is the new log message
                                      that is generated when the logout
                                      message is received. The macros
ending
                                      with @2 reference values of the
                                      previous message from the context. -
->                                   </values>
</message>                           </action>
</actions>               </rule>            </rules>
</ruleset></patterndb>
```

- It is possible to generate a message when the *context-timeout* of the original message expires and no new message is added to the context during this time. To accomplish this, include the **trigger="timeout"** attribute in the action element:

```
<action trigger="timeout">
```

Example 16.15. Sending alert when a client disappears

The following example shows how to combine various features of syslog-ng PE to send an e-mail alert if a client stops sending messages.

- Configure your clients to send MARK messages periodically. It is enough to configure MARK messages for the destination that forwards your log messages to your syslog-ng PE server (**mark-mode(periodical)**).
- On your syslog-ng PE server, create a pattern database rule that matches on the incoming MARK messages. In the rule, set the *context-scope* attribute to **host**, and the *context-timeout* attribute to a value that is higher than the *mark-freq* value set on your clients (by default, **mark-freq** is 1200 seconds, so set **context-timeout** at least to 1500

seconds, but you might want to use a higher value, depending on your environment).

- Add an action to this rule that sends you an e-mail alert if the **context-timeout** expires, and the server does not receive a new MARK message (**<action trigger="timeout">**).
- On your syslog-ng PE server, use the pattern database in the log path that handles incoming log messages.

Creating pattern databases

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, and
- a closing `@` character.

Example 16.16. 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::*@
```

Patterns and literals can be mixed together. For example, to parse a message that begins with the **`Host:`** string followed by an IP address (for example, **`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 16.17. 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: @"`**.

Pattern parsers of syslog-ng PE

The following parsers are available in syslog-ng PE.

@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::"@`**. You can use the colon (`:`) as stopcharacter as well, for example: **`@ESTRING:::@`**. You can also specify a stopstring instead of a single character, for example, **`@ESTRING::stop_here.@`**. The `@` character cannot be a stopcharacter, nor can line-breaks or tabs.

@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 (for example, 1, 0687, and so on). 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. A leading hyphen (–) is accepted for non-hexadecimal numbers, but other separator characters (for example, dot or comma) are not. To parse floating-point numbers, use the @FLOAT@ parser.

@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, for example: `@QSTRING::"@` parses everything between two quotation marks ("), while `@QSTRING:<>@` parses from an opening bracket to the closing bracket. The `@` character cannot be a quote character, nor can line-breaks or tabs.

@STRING@

A sequence of alphanumeric characters (0-9, A-z), not including any whitespace. Optionally, other accepted characters can be listed as parameters (for example, 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.

The syslog-ng pattern database format

Pattern databases are XML files that contain rules describing the message patterns. For sample pattern databases, see the section called “Downloading sample pattern databases”.

The following scheme describes the V4 format of the pattern database. This format is used by syslog-ng PE 4 F1 and later, and is backwards-compatible with the earlier V3 format.

For a sample database containing only a single pattern, see Example 16.18, “A V4 pattern database containing a single rule”.

TIP:

Use the **pdbtool** utility that is bundled with syslog-ng to test message patterns and convert existing databases to the latest format. For details, see pdbtool(1).

To automatically create an initial pattern database from an existing log file, use the **pdbtool patternize** command. For details, see the section called “The patternize command”.

Example 16.18. A V4 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='4' pub_date='2010-10-17'>    <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='4' pub_date='2010-10-17'>    <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>
```

Element: patterndb

Location

/**patterndb**

Description

The container element of the pattern database.

Attributes

- **version**: The schema version of the pattern database. The current version is `4`.
- **pubdate**: The publication date of the XML file.

Children

- **ruleset**

Example

```
<patterndb version='4' pub_date='2010-10-25'>
```

Element: ruleset

Location

/patterndb/**ruleset**

Description

A container element to group log patterns for an application or program. A **<patterndb>** element may contain any number of **<ruleset>** elements.

Attributes

- **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.

Children

- **patterns**
- **rules**
- **actions**
- **tags**

Example

```
<ruleset name='su' id='480de478-d4a6-4a7f-bea4-0c0245d361e1'>
```

Element: patterns

Location

/patterndb/ruleset/**patterns**

Description

A container element. A **<patterns>** element may contain any number of **<pattern>** elements.

Attributes

N/A

Children

- **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.

  Specifying multiple patterns is useful if two or more applications have different names (that is, different ${PROGRAM} fields), but otherwise send identical log messages.

  It is not necessary to use multiple patterns if only the end of the ${PROGRAM} fields is different, use only the beginning of the ${PROGRAM} field as the **pattern**. For example, the Postfix e-mail server sends messages using different process names, but all of them begin with the **postfix** string.

  > NOTE:
  >
  > If the **<pattern>** element of a ruleset is not specified, syslog-ng PE 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.

Example

```
<patterns>    <pattern>firstapplication</pattern>
<pattern>otherapplication</pattern></patterns>
```

Element: rules

Location

/patterndb/ruleset/**rules**

Description

A container element for the rules of the ruleset.

Attributes

N/A

Children

- **rule**

Example

```
<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>
```

Element: rule

Location

/patterndb/ruleset/rules/**rule**

Description

An element containing message patterns and how a message that matches these patterns is classified.

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.

Attributes

- **provider**: The provider of the rule. This is used to distinguish between who supplied the rule, that is, if it has been created by One Identity, 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.
-

context-id: OPTIONAL — An identifier to group related log messages when using the pattern database to correlate events. The ID can be a descriptive string describing the events related to the log message (for example, **`ssh-sessions`** for log messages related to SSH traffic), but can also contain macros to generate IDs dynamically. When using macros in IDs, see also the *`context-scope`* attribute. For details on correlating messages, see the section called "Correlating log messages".

> NOTE:
>
> The syslog-ng PE application determines the context of the message *after* the pattern matching is completed. This means that macros and name-value pairs created by the matching pattern database rule can be used as context-id macros.

- **context-timeout**: OPTIONAL — The number of seconds the context is stored. Note that for high-traffic log servers, storing open contexts for long time can require significant amount of memory. For details on correlating messages, see the section called "Correlating log messages".
- **context-scope**: OPTIONAL — Specifies which messages belong to the same context. This attribute is used to determine the context of the message if the *`context-id`* does not specify any macros. Usually, *`context-scope`* acts a filter for the context, with *`context-id`* refining the filtering if needed. The *`context-scope`* attribute has the following possible values:
  - *process*: Only messages that are generated by the same process of a client belong to the same context, that is, messages that have identical ${HOST}, ${PROGRAM} and ${PID} values. This is the default behavior of syslog-ng PE if *`context-scope`* is not specified.
  - *program*: Messages that are generated by the same application of a client belong to the same context, that is, messages that have identical ${HOST} and ${PROGRAM} values.
  - *host*: Every message generated by a client belongs to the same context, only the ${HOST} value of the messages must be identical.
  - *global*: Every message belongs to the same context.

  > NOTE:
  >
  > Using the *`context-scope`* attribute is significantly faster than using macros in the *`context-id`* attribute.

  For details on correlating messages, see the section called "Correlating log messages".

Children

- **patterns**

Example

```
<rule provider='balabit' id='f57196aa-75fd-11dd-9bba-001e6806451b'
class='violation'>
```

The following example specifies attributes for correlating messages as well. For details on correlating messages, see the section called "Correlating log messages".

```
<rule provider='balabit' id='f57196aa-75fd-11dd-9bba-001e6806451b' class='violation'
context-id='same-session' context-scope='process' context-timeout='360'>
```

Element: patterns

Location

/patterndb/ruleset/rules/rule/**patterns**

Description

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.

Attributes

N/A

Children

- **pattern**: A pattern describing a log message. This element is also called **`message pattern`**. For example:

  ```
  <pattern>+ ??? root-</pattern>
  ```

  > NOTE:
  >
  > Support for XML entities is limited, you can use only the following entities: **`& < > &quot; &apos;`**. User-defined entities are not supported.

- **description**: OPTIONAL — A description of the pattern or the log message matching the pattern.
- **urls**
- **values**

- **examples**

Example

```
<patterns>    <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @ for@QSTRING:SSH_
USERNAME: @from\ @QSTRING:SSH_CLIENT_ADDRESS: @port @NUMBER:SSH_PORT_NUMBER:@
ssh2</pattern></patterns>
```

Element: urls

Location

/patterndb/ruleset/rules/rule/patterns/**urls**

Description

OPTIONAL — An element containing one or more URLs referring to further information about the patterns or the matching log messages.

Attributes

N/A

Children

- **url**: OPTIONAL — An URL referring to further information about the patterns or the matching log messages.

Example

N/A

Element: values

Location

/patterndb/ruleset/rules/rule/patterns/**values**

Description

OPTIONAL — Name-value pairs that are assigned to messages matching the patterns, for example, the representation of the event in the message according to the Common Event Format (CEF) or Common Event Exchange (CEE). The names can be used as macros to reference the assigned values.

Attributes

N/A

Children

- **value**: OPTIONAL — Contains the value of the name-value pair that is assigned to the message.

  The <value> element of name-value pairs can include template functions. For details, see the section called "Using template functions", for examples, see the section called "if".

  When used together with message correlation, the <value> element of name-value pairs can include references to the values of earlier messages from the same context. For details, see the section called "Correlating log messages".
- *name*: The name of the name-value pair. It can also be used as a macro to reference the assigned value.

Example

```
<values>    <value name=".classifier.outcome">/Success</value></values>
```

Element: examples

Location

/patterndb/ruleset/rules/rule/patterns/**examples**

Description

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.

Attributes

N/A

Children

- **example**

Example

```
<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>
```

Element: example

Location

/patterndb/ruleset/rules/rule/patterns/examples/**example**

Description

OPTIONAL — A container element for a sample log message.

Attributes

N/A

Children

- **test_message**: OPTIONAL — A sample log message that should match this pattern. For example:

  ```
  <test_message program="myapplication">Content filter has been
  enabled</test_message>
  ```

  - *program*: The program pattern of the test message. For example:

```
<test_message program="proftpd">ubuntu (::ffff:192.168.2.179
[::ffff:192.168.2.179]) - FTP session closed.</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.

Example

```
<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>
```

Element: actions

Location

/patterndb/ruleset/**actions**

Description

OPTIONAL — A container element for actions that are performed if a message is recognized by the pattern. For details on actions, see the section called “Triggering actions for identified messages”.

Attributes

N/A

Children

- **action**

Example

Example 16.19. Generating messages for pattern database matches

When inserted in a pattern database rule, the following example generates a message when a message matching the rule is received.

```
<actions>    <action>        <message>            <values>
<value name="MESSAGE">A log message from ${HOST} matched rule number
$.classifier.rule_id</value>            </values>        </message>
</action></actions>
```

To inherit the properties and values of the triggering message, set the *inherit-properties* attribute of the *<message>* element to TRUE. That way the triggering log message is cloned, including name-value pairs and tags. If you set any values for the message in the *<action>* element, they will override the values of the original message.

Example 16.20. Generating messages with inherited values

The following action generates a message that is identical to the original message, but its $PROGRAM field is set to *overriding-original-program-name*

```
<actions>    <action>        <message inherit-properties='TRUE'>
<values>                <value name="PROGRAM">overriding-original-program-
name</value>            </values>        </message>    </action></actions>
```

Element: action

Location

/patterndb/ruleset/actions/**action**

Description

OPTIONAL — A container element describing an action that is performed when a message matching the rule is received.

Attributes

- *condition*: A syslog-ng filter expression. The action is performed only if the message matches the filter. The filter can include macros and name-value pairs extracted from the message.
- *rate*: Specifies maximum how many messages should be generated in the specified time period in the following format: **<number-of-messages>/<period-in-seconds>**. For example: **1/60** allows 1 message per minute. Rates apply within the scope of the context, that is, if **context-scope="host"** and **rate="1/60"**, then maximum one message is generated per minute for every host that sends a log message matching the rule. Excess messages are dropped. Note that when applying the rate to the generated messages, syslog-ng PE uses the timestamps of the log messages, similarly to calculating the *context-timeout*. That way **rate** is applied correctly even if the log messages are processed offline.
- *trigger*: Specifies when the action is executed. The *trigger* attribute has the following possible values:
  - *match*: Execute the action immediately when a message matching the rule is received.
  - *timeout*: Execute the action when the correlation timer (*context-timeout*) expires. This is available only if actions are used together with correlating messages.

Children

- **message**: A container element storing the message to be sent when the action is executed. Currently syslog-ng PE sends these messages to the *internal()* destination.
  - *inherit-properties*: If set to **TRUE**, the original message that triggered the action is cloned, including its name-value pairs and tags. For details, see the section called "Triggering actions for identified messages".

    If set to *context*, syslog-ng PE collects every name-value pair from each message stored in the context, and includes them in the generated message. If a name-value pair appears in multiple messages of the context, the value in the latest message will be used. Note that tags are not merged, the generated message will inherit the tags assigned to the last message of the context. For details on the message context, see the section called "Correlating log messages" and the section called "Actions and message correlation".

    This option is available in syslog-ng PE 5.3.2 and later.
- **values**: A container element for values and fields that are used to create the message generated by the action.

value: Sets the value of the message field specified in the *name* attribute of the element. For example, to specify the body of the generated message, use the following syntax:

```
<value name="MESSAGE">A log message matched rule number
$.classifier.rule_id</value>
```

Note that currently it is not possible to add DATE, FACILITY, or SEVERITY fields to the message.

When the action is used together with message correlation, the syslog-ng PE application automatically adds fields to the message based on the *context-scope* parameter. For example, using **context-scope="process"** automatically fills the HOST, PROGRAM, and PID fields of the generated message.

- *name*: Name of the message field set by the *value* element.

Example

Example 16.21. Generating messages for pattern database matches

When inserted in a pattern database rule, the following example generates a message when a message matching the rule is received.

```
<actions>    <action>        <message>            <values>
<value name="MESSAGE">A log message from ${HOST} matched rule number
$.classifier.rule_id</value>            </values>        </message>
</action></actions>
```

To inherit the properties and values of the triggering message, set the *inherit-properties* attribute of the *<message>* element to TRUE. That way the triggering log message is cloned, including name-value pairs and tags. If you set any values for the message in the *<action>* element, they will override the values of the original message.

Example 16.22. Generating messages with inherited values

The following action generates a message that is identical to the original message, but its $PROGRAM field is set to *overriding-original-program-name*

```
<actions>    <action>        <message inherit-properties='TRUE'>
<values>                <value name="PROGRAM">overriding-original-program-
name</value>            </values>        </message>    </action></actions>
```

Element: tags

Location

/patterndb/ruleset/**tags**

Description

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 (for details, see the section called "Tagging messages"). Starting with syslog-ng Premium Edition 3.2, the list of tags assigned to a message can be referenced with the `${TAGS}` macro.

Attributes

N/A

Children

- **tag**: OPTIONAL — A keyword or tags applied to messages matching the rule.

Example

```
<tags><tag>UserLogin</tag></tags>
```

Chapter 17. Statistics and metrics of syslog-ng

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:

- Using the **syslog-ng-ctl stats** command.
- Using the **syslog-ng-query sum "*"** command. Note that the output of the **syslog-ng-query** command is better structured, and also allows you to select which metrics you want to display. For details, see syslog-ng-query(1).
- 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 /opt/syslog-ng/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.

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. For details, see the section called “Global options”. 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, and so on). 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.)

     NOTE:

     The syslog-ng PE application stores the statistics of the objects when syslog-ng PE is reloaded. However, if the configuration of syslog-ng PE was changed since the last reload, the statistics of orphaned objects are deleted.
5. The type of the statistics:

- *processed*: The number of messages that successfully reached their destination driver. Note that this does not necessarily mean that the destination driver successfully delivered the messages (for example, written to disk or sent to a remote server).
- *dropped*: The number of dropped messages — syslog-ng PE could not send the messages to the destination and the output buffer got full, so messages were dropped by the destination driver.
- *stored*: The number of messages stored in the message queue of the destination driver, 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:
>
> Certain statistics are available only if the *stats-level()* option is set to a higher value.
>
> When receiving messages with non-standard facility values (that is, higher than 23), these messages will be listed as *other* facility instead of their facility number.

To reset the statistics to zero, use the following command: **syslog-ng-ctl stats --reset**

Starting with version 4 F1, syslog-ng PE can process sources and destinations in multithreaded mode to scale to multiple CPUs or cores for increased performance. Starting with version 5 F4, this multithreaded mode is the default.

Multithreading concepts of syslog-ng PE

This section is a brief overview on how syslog-ng PE works in multithreaded mode. It is mainly for illustration purposes: the concept has been somewhat simplified and may not completely match reality.

> **NOTE:**
>
> The way syslog-ng PE uses multithreading may change in future releases. The current documentation applies to version 6 LTS.

syslog-ng PE always uses multiple threads:

- A main thread that is always running
- A number of worker threads that process the messages. You can influence the behavior of worker threads using the *threaded()* option and the *--worker-threads* command-line option.
- Some other, special threads for internal functionalities. For example, certain destinations run in a separate thread, independently of the multithreading (*threaded()*) and *--worker-threads* settings of syslog-ng PE.

The maximum number of worker threads syslog-ng PE uses is the number of CPUs or cores in the host running syslog-ng PE (up to 64). You can limit this value using the *--worker-threads* command-line option that sets the maximum total number of threads syslog-ng PE can use, including the main syslog-ng PE thread. However, the *--worker-threads* option does not affect the supervisor of syslog-ng PE. The supervisor is a separate process (see syslog-ng(8)), but certain operating systems might display it as a thread. In addition, certain destinations always run in a separate thread, independently of the multithreading (*threaded()*) and *--worker-threads* settings of syslog-ng PE.

When an event requiring a new thread occurs (for example, syslog-ng PE receives new messages, or a destination becomes available), syslog-ng PE tries to start a new thread. If there are no free threads, the task waits until a thread finishes its task and becomes available. There are two types of worker threads:

- Reader threads read messages from a source (as many as possible, but limited by the *log-fetch-limit()* and *log-iw-size()* options. The thread then processes these messages, that is, performs filtering, rewriting and other tasks as necessary, and puts the log message into the queue of the destination. If the destination does

not have a queue (for example, usertty), the reader thread sends the message to the destination, without the interaction of a separate writer thread.

- Writer threads take the messages from the queue of the destination and send them to the destination, that is, write the messages into a file, or send them to the syslog server over the network. The writer thread starts to process messages from the queue only if the destination is writable, and there are enough messages in the queue, as set in the *flush-lines()* and the *flush-timeout()* options. Writer threads stop processing messages when the destination becomes unavailable, or there are no more messages in the queue.

Sources and destinations affected by multithreading. The following list describes which sources and destinations can use multiple threads. Changing the *--worker-threads* command-line option changes the number of threads available to these sources and destinations.

- The *tcp* and **syslog(tcp)** sources can process independent connections in separate threads. The number of independent connections is limited by the *max-connections()* option of the source. Separate sources are processed by separate thread, for example, if you have two separate *tcp* sources defined that receive messages on different IP addresses or port, syslog-ng PE will use separate threads for these sources even if they both have only a single active connection.
- The *udp*, *file*, and *pipe* sources use a single thread for every source statement.
- The *tcp*, *syslog*, and *pipe* destinations use a single thread for every destination.
- The *file* destination uses a single thread for writing the destination file, but may use a separate thread for each destination file if the filename includes macros.

Sources and destinations not affected by multithreading. The following list describes sources and destinations that use a separate thread even if you disable multithreading in syslog-ng PE, in addition to the limit set in the *--worker-threads* command-line option.

- The *logstore* destination uses separate threads for writing the messages from the journal to the logstore files, and also for timestamping. These threads are independent from the setting of the *--worker-threads* command-line option.
- Every *sql* destination uses its own thread. These threads are independent from the setting of the *--worker-threads* command-line option.
- The *java* destinations use one thread, even if there are multiple Java-based destinations configured. This thread is independent from the setting of the *--worker-threads* command-line option.

Configuring multithreading

Starting with version 5 F4, syslog-ng PE runs in multithreaded mode by default. You can enable multithreading in syslog-ng PE using the following methods:

- Globally using the **`threaded(yes)`** option.
- Separately for selected sources or destinations using the **`flags("threaded")`** option.

Example 18.1. Enabling multithreading

To enable multithreading globally, use the *`threaded`* option:

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

To enable multithreading only for a selected source or destination, use the **`flags("threaded")`** option:

```
source s_tcp_syslog { network(ip(127.0.0.1) port(1999) flags("syslog-
protocol", "threaded") ); };
```

Optimizing multithreaded performance

Sources:

In syslog-ng, every source has a reader thread. To improve scaling on the source side, use multiple sources instead of one.

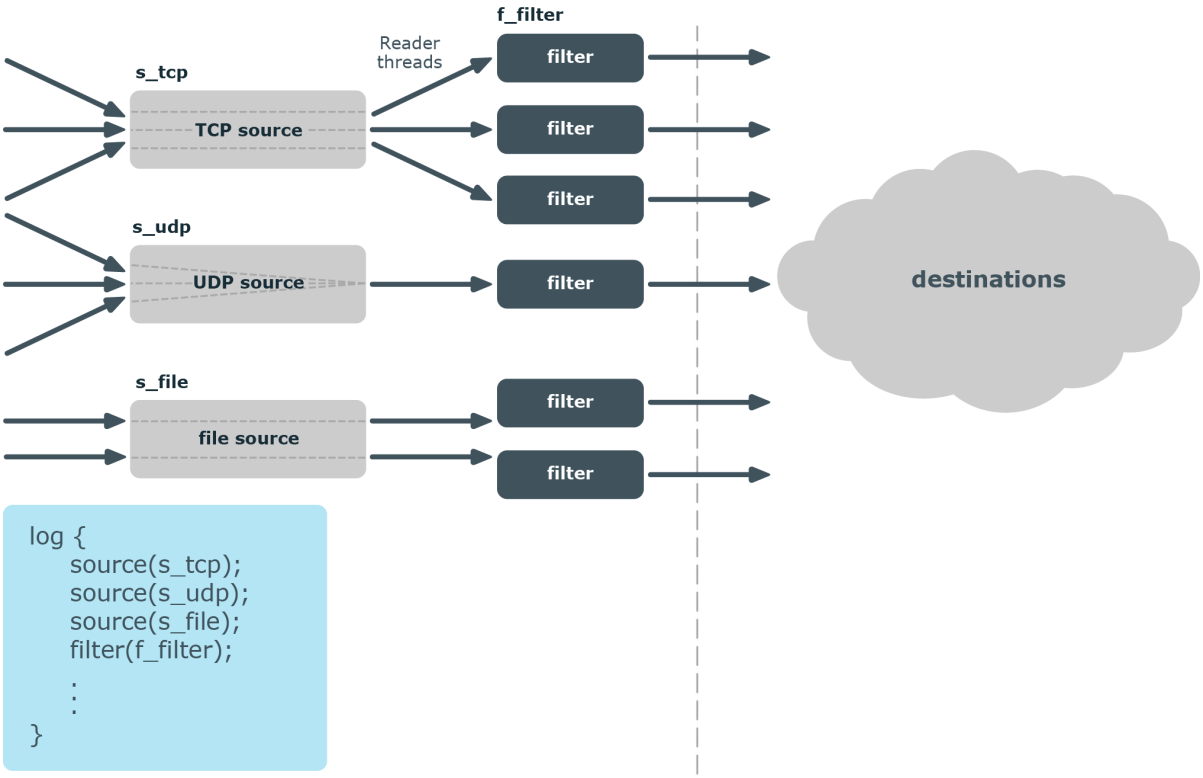
For example, if you are using a file source with a wildcard character (such as `*.txt`), syslog-ng will monitor every matching file (and keep switching between them), but will use only one thread. It is best to configure several single file sources (such as file source1, file source2, and so on) that all monitor only a single file or a smaller range of files. In this case, every file source will use its own thread.

TCP-based network sources form an exception: a TCP-based network source will scale based on the number of active connections. This means that if there are 10 incoming TCP connections all coming to the same network source, then that source can use 10 threads, one thread for each connection.

NOTE:

UDP-based network sources do not scale by themselves because they always use a single thread. If you want to handle a large number of UDP connections, it is best to configure a subset of your clients to send the messages to a different port of your syslog-ng server, and use separate source definitions for each port.

Figure 18.1. How multithreading works — sources



Message processors:

Message processors — such as filters, rewrite rules, and parsers — are executed by the reader thread in a sequential manner.

For example, if you have a log path that defines two sources and a filter, the filter will be performed by the source1 reader thread when log messages come from source1, and by the source2 reader thread when log messages come from source2. This means that if log messages come from both source1 and source2, they will both have a reader thread and that way filtering will be performed simultaneously.

NOTE:

This is not true for PatternDB because it uses message correlation. When using PatternDB, it runs in only one thread at a time, and this significantly decreases performance.

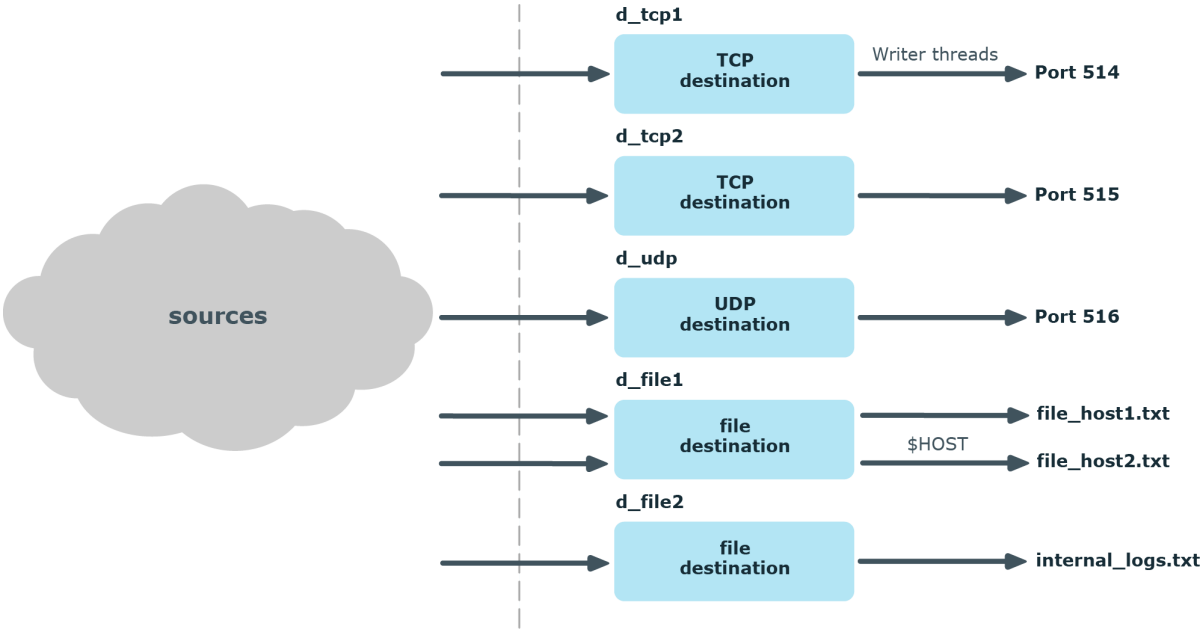
Destinations:

In syslog-ng, every destination has a writer thread. To improve scaling on the destination side, use multiple destinations instead of one.

For example, when sending messages to a syslog-ng server, you can use multiple connections to the server if you configure the syslog-ng server to receive messages on multiple ports, and configure the clients to use both ports.

When writing the log messages to files, use macros in the filename to split the messages to separate files (for example, using the ${HOST} macro). Files with macros in their filenames are processed in separate writer threads.

Figure 18.2. How multithreading works — destinations



This chapter provides tips and guidelines about troubleshooting problems related to syslog-ng. Troubleshooting the syslog-ng Agent for Windows application is discussed in *Administration Guide for syslog-ng Agent for Windows*.

- 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.
- Always check the configuration files for any syntax errors on both the client and the server using the **syslog-ng --syntax-only** command.
- If the syslog-ng PE server does not receive the messages, verify that the IP addresses and ports are correct in your sources and destinations. Also, check that the client and the server uses the same protocol (a common error is to send logs on UDP, but configure the server to receive logs on TCP.

  If the problem persist, 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.
- To find message-routing problems, run syslog-ng PE with the following command **syslog-ng -Fevd**. That way syslog-ng PE will run in the foreground, and display debug messages about the messages that are processed.
- 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. You can enable these messages without restarting syslog-ng using the **syslog-ng-ctl verbose --set=on** command. For details, see the syslog-ng-ctl man page at syslog-ng-ctl(1).
- 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.
- If you use the same driver and options in the destination of your syslog-ng PE client and the source of your syslog-ng PE server, everything should work as expected. Unfortunately there are some other combinations, that seem to be working, but result in losing parts of the messages. For details on the working combinations, see the section called “Things to consider when forwarding messages between syslog-ng PE hosts”.

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 (for details, see the section called "Managing incoming and outgoing messages with flow-control").

If your syslog-ng PE host uses an NFS partition, see the section called "NFS file system for log files".

- *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. Use flags (flow-control) to avoid it (for details, see the section called "Configuring flow-control"). The number of dropped messages is displayed per destination in the log message statistics of syslog-ng (for details, see Chapter 17, *Statistics and metrics of syslog-ng*).
- *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.

  For details on minimizing message loss when using UDP, see the following tutorial: *Collecting log messages from UDP sources*.
- *In the socket receive buffer*: When transferring messages using the UDP protocol, the UDP datagram (that is, 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 (for details, see Chapter 17, *Statistics and metrics of syslog-ng*).
  - If the number of Log Source Hosts reaches the license limit, the syslog-ng PE server will not accept connections from additional hosts. The messages sent by additional hosts will be dropped, even if the client uses a reliable transport method (for example, RLTP).
- *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. For details, see the previous entry. 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.
- *When syslog-ng PE is writing messages into files*: If syslog-ng PE receives a signal (SIG) while writing log messages to file, the log message that is processed by the *write* call can be lost if the *flush_lines* parameter is higher than 1.
- *When using the RLTP™ protocol*:

  In the following cases, it is possible to lose log messages even if you use RLTP™:

  - If you use RLTP™ together with non-reliable disk-buffer, it is possible to lose logs.
  - When sending logs through a relay that is using a non-reliable disk-buffer, it is possible to lose logs if the relay crashes.
  - When sending logs through a relay that is using a non-reliable disk-buffer, it is possible that logs are duplicated if the relay crashes, or it is stopped.
  - If the underlying disk system of syslog-ng PE fails to write the log messages to the disk, but it does not return a write error, or some other hardware or operating-system error happens.

    NOTE:

    To minimize the chances of losing messages, it is recommended to set the *flush-size()* parameter of the receiver to 1 (however, note that depending on your syslog-ng PE configuration, this can decrease the performance of the receiver).

Procedure 19.1. Creating syslog-ng core files

Purpose:

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:

Steps:

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
   ```

2. Verify that syslog-ng has permissions to write the directory it is started from, for example `/opt/syslog-ng/sbin/`.
3. If syslog-ng crashes, it will create a core file in the directory syslog-ng was started from.

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.

Procedure 19.2. Running a failure script

Purpose:

You can create a failure script that is executed when syslog-ng PE terminates abnormally, that is, when it exits with a non-zero exit code. For example, you can use this script to send an automatic e-mail notification.

Prerequisites:

The failure script must be the following file: `/opt/syslog-ng/sbin/syslog-ng-failure`, and must be executable.

To create a sample failure script, complete the following steps.

Steps:

1. Create a file named `/opt/syslog-ng/sbin/syslog-ng-failure` with the following content:

```
#!/usr/bin/env bash
cat >>/tmp/test.txt <<EOF
$(date)
Name............$1
Chroot dir......$2
Pid file dir....$3
Pid file........$4
Cwd.............$5
Caps............$6
Reason..........$7
Argbuf..........$8
Restarting......$9

EOF
```

2. Make the file executable: **chmod +x /opt/syslog-ng/sbin/syslog-ng-failure**
3. Run the following command in the `/opt/syslog-ng/sbin` directory: **./syslog-ng --process-mode=safe-background; sleep 0.5; ps aux | grep './syslog-ng' | grep -v grep | awk '{print $2}' | xargs kill -KILL; sleep 0.5; cat /tmp/test.txt**

   The command starts syslog-ng PE in safe-background mode (which is needed to use the failure script) and then kills it. You should see that the relevant information is written into the `/tmp/test.txt` file, for example:

```
Thu May 18 12:08:58 UTC 2017
Name............syslog-ng
Chroot dir......NULL
Pid file dir....NULL
Pid file........NULL
Cwd.............NULL
Caps............NULL
Reason..........signalled
Argbuf..........9
Restarting......not-restarting
```

4. You should also see messages similar to the following in system syslog. The exact message depends on the signal (or the reason why syslog-ng PE stopped):

   ```
   May 18 13:56:09 myhost supervise/syslog-ng[10820]: Daemon exited gracefully,
   not restarting; exitcode='0'
   May 18 13:57:01 myhost supervise/syslog-ng[10996]: Daemon exited due to a
   deadlock/signal/failure, restarting; exitcode='131'
   May 18 13:57:37 myhost supervise/syslog-ng[11480]: Daemon was killed, not
   restarting; exitcode='9'
   ```

   The failure script should run on every non-zero exit event.

Collecting debugging information with strace, truss, or tusc

To properly troubleshoot certain situations, it can be useful to trace which system calls syslog-ng PE performs. How this is performed depends on the platform running syslog-ng PE. In general, note the following points:

- When syslog-ng PE is started, a supervisor process might stay in the foreground, while the actual syslog-ng daemon goes to the background. Always trace the background process.
- Apart from the system calls, the time between two system calls can be important as well. Make sure that your tracing tool records the time information as well. For details on how to do that, refer to the manual page of your specific tool (for example, **strace** on Linux, or **truss** on Solaris and BSD).
- Run your tracing tool in verbose mode, and if possible, set it to print long output strings, so the messages are not truncated.
- When using **strace**, also record the output of **lsof** to see which files are accessed.

The following are examples for tracing system calls of syslog-ng on some platforms. The output is saved into the `/tmp/syslog-ng-trace.txt` file, sufficed with the PID of the related syslog-ng process.The path of the syslog-ng binary assumes that you have installed syslog-ng PE from the official syslog-ng PE binaries available at the BalaBit website — native distribution-specific packages may use different paths.

- *Linux*: **strace -o /tmp/trace.txt -s256 -ff -ttT /opt/syslog-ng/sbin/syslog-ng -f /opt/syslog-ng/etc/syslog-ng.conf -Fdv**
- *HP-UX*: **tusc -f -o /tmp/syslog-ng-trace.txt -T /opt/syslog-ng/sbin/syslog-ng**
- *IBM AIX and Solaris*: **truss -f -o /tmp/syslog-ng-trace.txt -r all -w all -u libc:: /opt/syslog-ng/sbin/syslog-ng -d -d**

TIP:

To execute these commands on an already running syslog-ng PE process, use the **-p <pid_of_syslog-ng>** parameter.

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**.

Reporting bugs and finding help

If you need help, want to open a support ticket, or report a bug, we recommend using the **syslog-debun** tool to collect information about your environment and syslog-ng PE version. For details, see syslog-debun(1). For support, contact our Support Team.

Recover data from orphaned diskbuffer files

When you change the configuration of a syslog-ng PE host that uses disk-based buffering (also called disk queue), syslog-ng PE may start new disk buffer files for the destinations that you have changed. In such case, syslog-ng PE abandons the old disk queue files. If there were unsent log messages in the disk queue files, these messages remain in the disk queue files, and will not be sent to the destinations.

To find, examine, and flush the log messages from such orphaned disk queue files, see the *Sending out messages stuck in syslog-ng disk queue files* tutorial.

This chapter discusses some special examples and recommendations.

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:

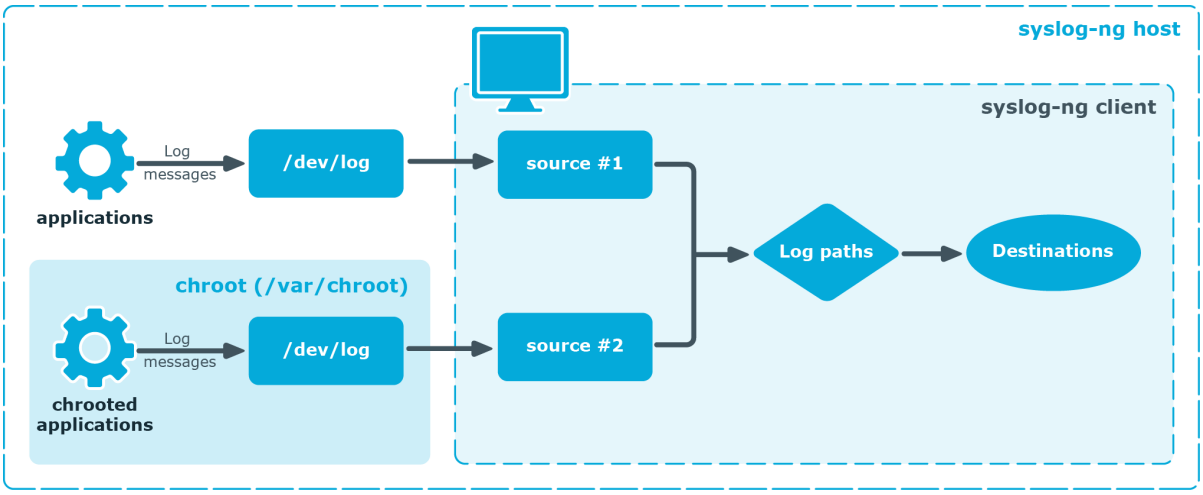
  ```
  options {ts-format(iso) ; };
  ```
- Resolving the IP addresses of the clients to domain names can decrease the performance of syslog-ng. For details, see the section called “Using name resolution in syslog-ng”.

Procedure 20.2. Collecting logs from chroot

Purpose:

To collect logs from a chroot using a syslog-ng client running on the host, complete the following steps:

Figure 20.1. Collecting logs from chroot



Steps:

1. Create a `/dev` directory within the chroot. The applications running in the chroot send their log messages here.
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).
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`.

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. For details, see the section called "Using name resolution in syslog-ng".
- Enable flow-control for the TCP sources. For details, see the section called "Managing incoming and outgoing messages with flow-control".
- 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. For details, see the section called "Regular expressions".

  CAUTION:

  When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng PE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the *so-rcvbuf()* option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

  As a general rule, increase the *so-rcvbuf()* so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the *so-rcvbuf()* at least to `2 097 152` bytes.
- Increase the value of the *flush-lines()* parameter. Increasing *flush-lines()* from `0` to `100` can increase the performance of syslog-ng PE by 100%.

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-size(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. For details, see Procedure 20.1, "Resolving hostnames locally".

NOTE:

Domain name resolution is important mainly in relay and server mode.

Procedure 20.1. Resolving hostnames locally

Purpose:

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:

Steps:

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.
2. Instruct syslog-ng to resolve hostnames locally. Set the *`use-dns()`* option of syslog-ng to *`persist_only`*.
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); };
   ```

Configuring log rotation

The syslog-ng PE application does not rotate logs by itself. To use syslog-ng PE for log rotation, consider the following approaches:

Use logrotate together with syslog-ng PE:

- Ideal for workstations or when processing fewer logs.
- It is included in most distributions by default.
- Less scripting is required, only **logrotate** has to be configured correctly.
- Requires frequent restart (syslog-ng PE must be reloaded/restarted when the files are rotated). After rotating the log files, reload syslog-ng PE using the **syslog-ng-ctl reload** command, or use another method to send a SIGHUP to syslog-ng PE.
- The statistics collected by syslog-ng PE, and the correlation information gathered with Pattern Database is lost with each restart.

Separate incoming logs based on time, host or other information:

- Ideal for central log servers, where regular restart of syslog-ng PE is unfavorable.
- Requires shell scripts or cron jobs to remove old logs.
- It can be done by using macros in the destination name (in the filename, directory name, or the database table name).

Example 20.1. File destination for log rotation

This sample file destination configuration stores incoming logs in files that are named based on the current year, month and day, and places these files in directories that are named based on the hostname:

```
destination d_sorted { file("/var/log/remote/${HOST}/${YEAR}_${MONTH}_
${DAY}.log" create-dirs(yes)); };
```

Example 20.2. Logstore destination for log rotation

This sample logstore destination configuration stores incoming logs in logstores that are named based on the current year, month and day, and places these logstores in directories that are named based on the hostname:

```
destination d_logstore { logstore("/var/log/remote/${HOST}/${YEAR}_${MONTH}_
${DAY}.lgs" compress(9) create-dirs(yes)); };
```

Example 20.3. Command for cron for log rotation

This sample command for **cron** removes files older than two weeks from the `/var/log/remote` directory:

```
find /var/log/remote/ -daystart -mtime +14 -type f -exec rm {} \;
```

About us

One Identity solutions eliminate the complexities and time-consuming processes often required to govern identities, manage privileged accounts and control access. Our solutions enhance business agility while addressing your IAM challenges with on-premises, cloud and hybrid environments.

Contacting us

For sales and other inquiries, such as licensing, support, and renewals, visit https://www.oneidentity.com/company/contact-us.aspx.

Technical support resources

Technical support is available to One Identity customers with a valid maintenance contract and customers who have trial versions. You can access the Support Portal at https://support.oneidentity.com/.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to videos at www.YouTube.com/OneIdentity
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product