syslog-ng Premium Edition 7.0.29

Administration Guide

Legend

WARNING: A WARNING icon highlights a potential risk of bodily injury or property damage, for which industry-standard safety precautions are advised. This icon is often associated with electrical hazards related to hardware.

CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.

syslog-ng PE Administration Guide
Updated - 14 January 2022, 01:10
Version - 7.0.29

Contents

1

Preface

Welcome to the syslog-ng Premium Edition 7 Administration Guide.

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

Summary of contents

Introduction to syslog-ng describes the main functionality and purpose of syslog-ng PE.

The concepts of syslog-ng discusses the technical concepts and philosophies behind syslog-ng PE.

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

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

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.

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

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

Routing messages: log paths, flags, and filters explains how to route and sort log messages, and how to use filters to select specific messages.

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

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

Advanced Log Transfer Protocol shows how to log using the Advanced Log Transfer Protocol protocol.

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

parser: Parse and segment structured messages describes how to segment and process structured messages like comma-separated values.

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

Correlating log messages details how to correlate log messages.

Enriching log messages with external data details how to import data from external sources to include in the log messages, thus extending, enriching, and complementing the data found in the log message.

Monitoring statistics and metrics of syslog-ng details the available statistics that syslog-ng PE collects about the processed log messages.

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

Troubleshooting syslog-ng offers tips to solving problems.

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

The syslog-ng manual pages contains the manual pages of the syslog-ng PE application.

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.

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.

2

Introduction to syslog-ng

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.

NOTE: Due to complexity of deployment, configuration, and design, you may require assistance from One Identity Professional Services while introducing new or additional:

- sources
- destinations
- log paths
- significant increases in log volume.

One Identity Professional Services is equipped and trained to evaluate the needs of any organization, and to provide configuration and architectural recommendations that help our users get the most out of any syslog-ng PE version.

One Identity Professional Services offer assistance in planning and scoping for current needs, as well as recommendations for the future to ensure success.

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 ALTPTCP protocol ensures that no messages are lost.

The disk-buffer option for messages

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 option 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.

Big data clusters

The log data that your organization has to process, store, and review increases daily, so many organizations use big data solutions for their logs. To accomodate this huge amount of data, syslog-ng PE natively supports storing log messages in HDFS files and Elasticsearch clusters.

Message queue support

Large organizations increasingly rely on queuing infrastructure to transfer their data. For that purpose, syslog-ng PE supports Apache Kafka, the Advanced Message Queuing Protocol (AMQP), and the Simple Text Oriented Messaging Protocol (STOMP).

SQL, NoSQL, and monitoring

Storing your log messages in a database allows you to easily search and query the messages and interoperate with log analyzing applications. The syslog-ng application supports the following databases: MongoDB, MSSQL, MySQL, Oracle, PostgreSQL, and SQLite.

syslog-ng PE also allows you to extract the information you need from your log data, and directly send it to your Graphite, Redis, or Riemann monitoring system.

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.

Heterogeneous environments

The syslog-ng PE application is the ideal choice to collect logs in massively heterogeneous environments using several different operating systems and hardware platforms, including Linux, Unix, BSD, Sun Solaris, HP-UX, and AIX.

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.

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 635,000 messages per second (over 235 MB of data per second) when receiving messages from multiple connections and storing them in text files.

- Over 615,000 messages per second (over 230 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 7?

- For details on the news and highlights of syslog-ng Premium Edition 7, see the Release Notes.
- For details on changes in The syslog-ng Premium Edition 7 Administrator Guide, see Summary of changes.

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.

Supported platforms

The syslog-ng Premium Edition application is officially supported on the following platforms. Note that the following table is for general reference only, and is not always accurate about the supported platforms and options available for specific platforms. Unless explicitly noted otherwise, the subsequent releases of the platform (for example, Red Hat Enterprise Linux 7 and its update releases like 7.2) are also supported.

For the x86_64 architecture, the following platforms are supported in version 7.0.29 of syslog-ng PE:

- CentOS 7
- Debian 10 (Buster)
- Oracle Linux 7
- Red Hat EL 7
- Red Hat EL 8
- SLES 12
- SLES 15
- Ubuntu 16.04 LTS (Xenial Xerus)
- Ubuntu 18.04 LTS (Bionic Beaver)
- Ubuntu 20.04 LTS (Focal Fossa)

For details about the syslog-ng Agent for Windows application, see the syslog-ng Agent for Windows documentation.

For using syslog-ng PE on other platforms (for example, AIX), see the list of supported platforms in the syslog-ng PE version 6 Administration Guide.

Certified packages

Starting from version 4.0, syslog-ng Premium Edition is *Novell Ready* certified for the following platforms:

- SUSE Linux Enterprise Server 10 on the x86 and x86_64 AMD64 & Intel EM64T architectures
- SUSE Linux Enterprise Server 11 on the x86 and x86_64 AMD64 & Intel EM64T architectures

Starting from version 4.0, syslog-ng Premium Edition is *RedHat Ready* certified for the following platforms:

- Red Hat Enterprise Linux 2.1 on the x86 architecture
- Red Hat Enterprise Linux 3 on the x86_64 AMD64 & Intel EM64T architecture
- Red Hat Enterprise Linux 4 on the x86 and x86_64 AMD64 & Intel EM64T architectures
- Red Hat Enterprise Linux 5 on the x86 and x86_64 AMD64 & Intel EM64T architectures
- Red Hat Enterprise Linux 6 on the x86 and x86_64 AMD64 & Intel EM64T architectures

Starting from version 5.4, syslog-ng Premium Edition is *MapR* certified.

3

The concepts of syslog-ng

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.

NOTE: Due to complexity of deployment, configuration, and design, you may require assistance from One Identity Professional Services while introducing new or additional:

- sources
- destinations
- log paths
- significant increases in log volume.

One Identity Professional Services is equipped and trained to evaluate the needs of any organization, and to provide configuration and architectural recommendations that help our users get the most out of any syslog-ng PE version.

One Identity Professional Services offer assistance in planning and scoping for current needs, as well as recommendations for the future to ensure success.

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.

NOTE: Due to complexity of deployment, configuration, and design, you may require assistance from One Identity Professional Services while introducing new or additional:

- sources
- destinations
- log paths
- significant increases in log volume.

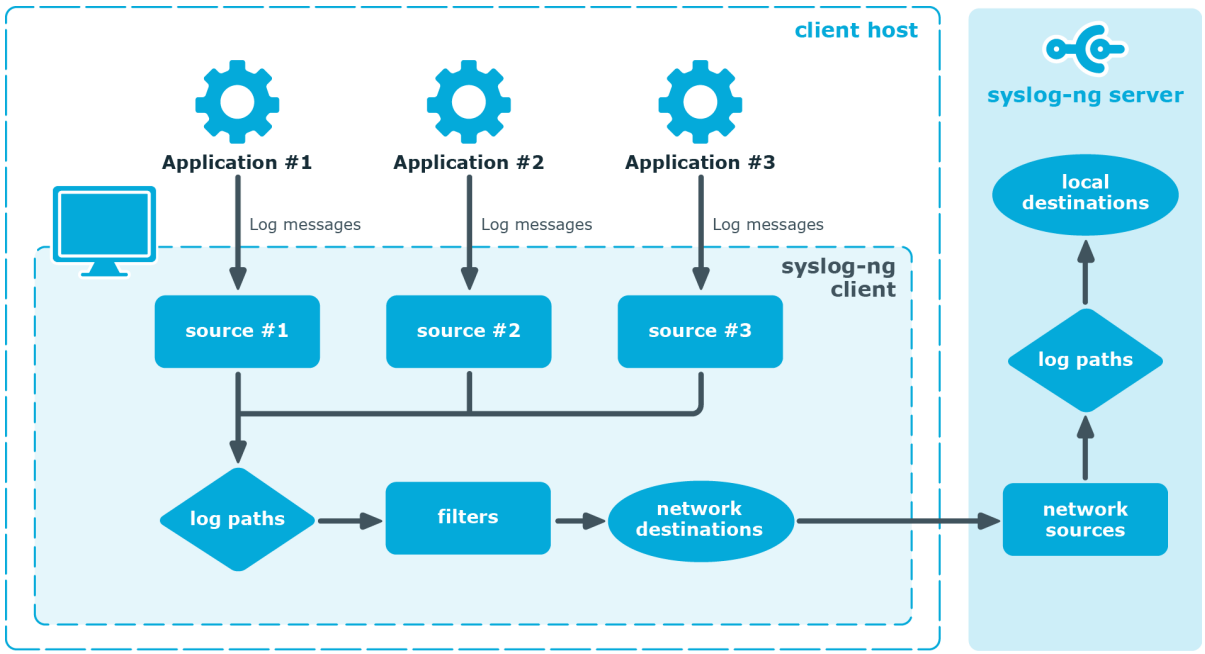
One Identity Professional Services is equipped and trained to evaluate the needs of any organization, and to provide configuration and architectural recommendations that help our users get the most out of any syslog-ng PE version.

One Identity Professional Services offer assistance in planning and scoping for current needs, as well as recommendations for the future to ensure success.

The route of a log message in syslog-ng

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 1: The route of a log message



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 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.

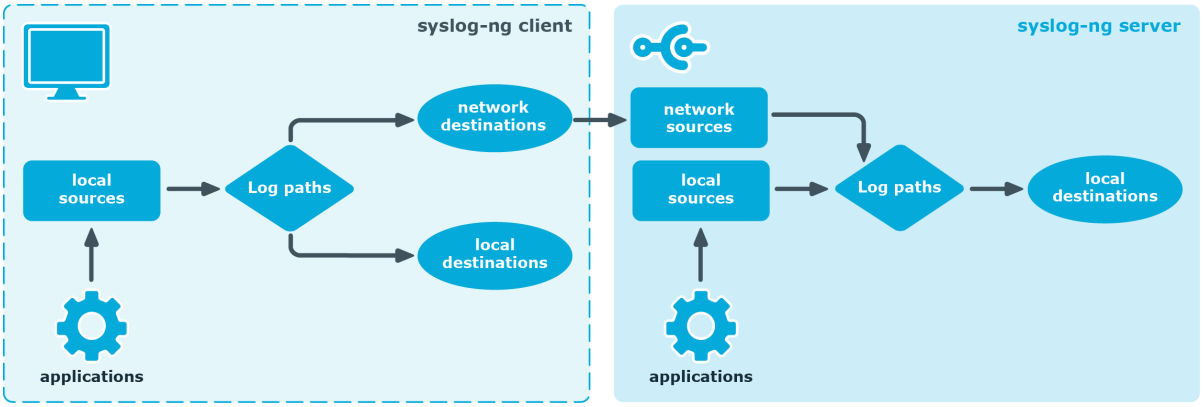
   NOTE: The syslog-ng application can stop reading messages from its sources if the destinations cannot process the sent messages. This feature is called flow-control and is detailed in Managing incoming and outgoing messages with flow-control.

Modes of operation

The syslog-ng Premium Edition application has three distinct typical 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: Client-mode operation

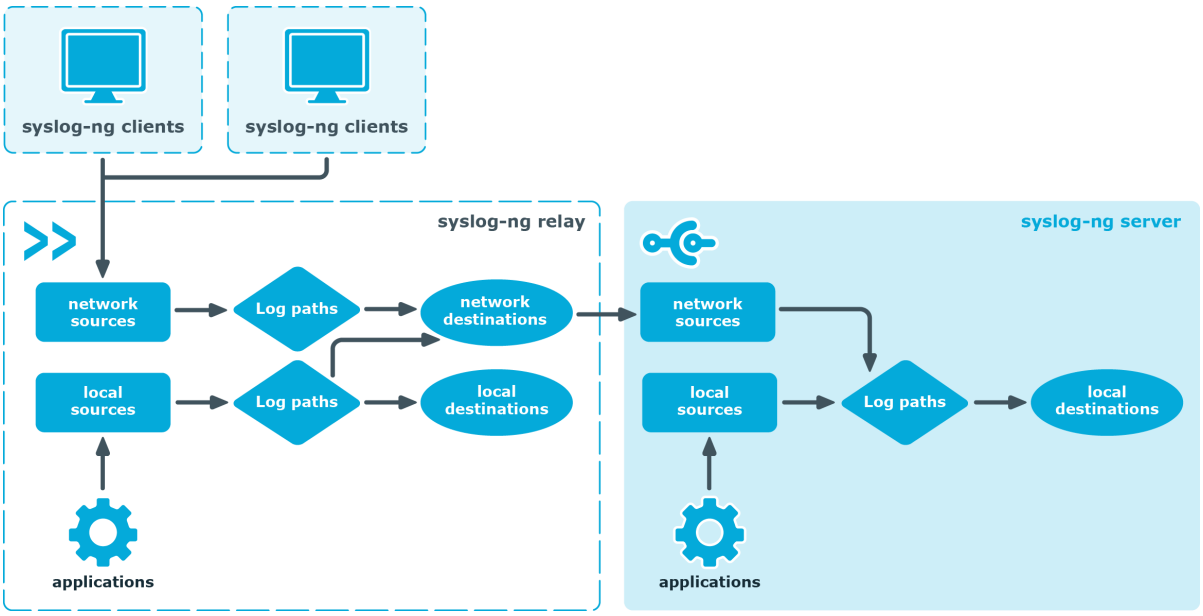


In client mode, syslog-ng collects the local logs generated by the host and forwards them through a network connection to the central syslog-ng server or to a relay. Clients often also log the messages locally into files.

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

Relay mode

Figure 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: `elasticsearch2()`, `hdfs()`, `kafka()`, `mongodb()`, `pipe()`, `smtp()`, `sql()` and `stackdriver()`. 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.

Example relay use cases

The relay collects log messages through the network and after processing, but without writing them on the disk for storage, forwards them to one or more remote destinations.

You can use a relay for many different use cases as described in the examples below.

UDP-only source devices

Most network devices send log messages over UDP. However, UDP does not guarantee that all packets are delivered, which makes UDP unreliable.

To ensure at least a best effort level of reliability, One Identity recommends that you deploy a relay on the network, close to the source devices. With the most reliable hops between the source and the relay, you can minimize the risk of losing UDP packets. Once the packet arrives at the relay, syslog-ng OSEsyslog-ng PE ensures that the messages are delivered to the central server in a reliable manner, based on TCP/TLS and Advanced Log Transfer Protocol (ALTP).

Too many source devices

Depending on the hardware and configuration, an average syslog-ng instance can usually handle the following number of concurrent connections:

- If the maximum message rate is lower than 200,000 messages per second:
  - maximum ca. 5,000 TCP connections
  - maximum ca. 1,000 TLS connections
  - maximum ca. 1,000 ALTP connections
- If the message rate is higher than 200,000 messages per second, contact One Identity.

If you have more source devices, you must deploy a relay machine at least per 5,000 sources and batch up all the logs into a single TCP connection that connects the relay to the server. If TLS or ALTP is used, deploy relays per 1,000 source devices.

Collecting logs from remote sites (especially over public WAN)

If you need to collect log messages from geographically remote sites or over public WAN, One Identity recommends that you install at least a relay node per each remote site. The relay can be the last outgoing hop for all the messages of the remote site, which has several benefits:

- **Maintenance**: You only need to change the configuration of the relay if you want to re-route the logs of some or all sources of the remote site. Also you do not need to change each source's configuration one by one.
- **Security**: If you trust your internal network, it is not necessary to hold encrypted connections within the LAN of the remote site as the messages can get to the relay without encryption. Messages must be sent in an encrypted way over the public WAN, and it is enough to hold only a single TCP/TLS connection between the sites, that is, between the remote relay and the central server. This eliminates the wasting of resources as holding several TLS connections directly from the clients is more costly than holding a single connection from the relay.
- **Reliability**: You can set up a main disk-buffer on the relay. The main disk-buffer is only responsible for buffering all the logs of the remote site if the central syslog-ng OSEsyslog-ng PE server is temporarily unavailable. It is easier to maintain this single main disk-buffer instead of setting disk-buffers on individual client machines.

Separation, distribution, and balancing of message processing tasks

Most Linux applications have their own human readable, but difficult to handle, log messages. Without parsing and normalization it is difficult to alert and report on these log messages. Many syslog-ng users use the message parsing tools of syslog-ng to normalize their different log messages. Just like normalization, filtering can also be resource-heavy, depending on what the filtering is based on. In this case, it might be inefficient to perform all the message processing tasks on the server as it can result in decreased overall performance.

It is a typical setup to deploy relays in front of the central server operating as a receiver front-end. Most resource-heavy tasks, for example, parsing, filtering, and so on, are performed on this receiver layer. As all resource-heavy tasks are performed on the relay, the central server behind it only needs to get the messages from the relay and write them into the final text-based or tamper-proof (logstore) format. Since you can run several relays, you can balance the resource-heavy tasks between more relays, and a single server behind the relays can still be fast enough to write all the messages on the disk.

Acting as a relay also depends on the functionality. A relay does not have to be a dedicated relay machine at all. For log collection, it can be one of the clients with a relay configuration. Note that in a robust log collection infrastructure, the relays have their own purpose, and One Identity recommends running dedicated relay machines.

You can run several parallel relays to ensure horizontal redundancy. For example, if each of the relays has the same configuration, when one relay goes down another relay can take over the processing. Distribution of the logs can be done by the built-in client-side failover functionality and also by a general load balancer. The load balancer is also used to serve N+1 redundant relay deployments. In this case, switching from one relay to another relay is done when there is an outage but also for real load balancing purposes.

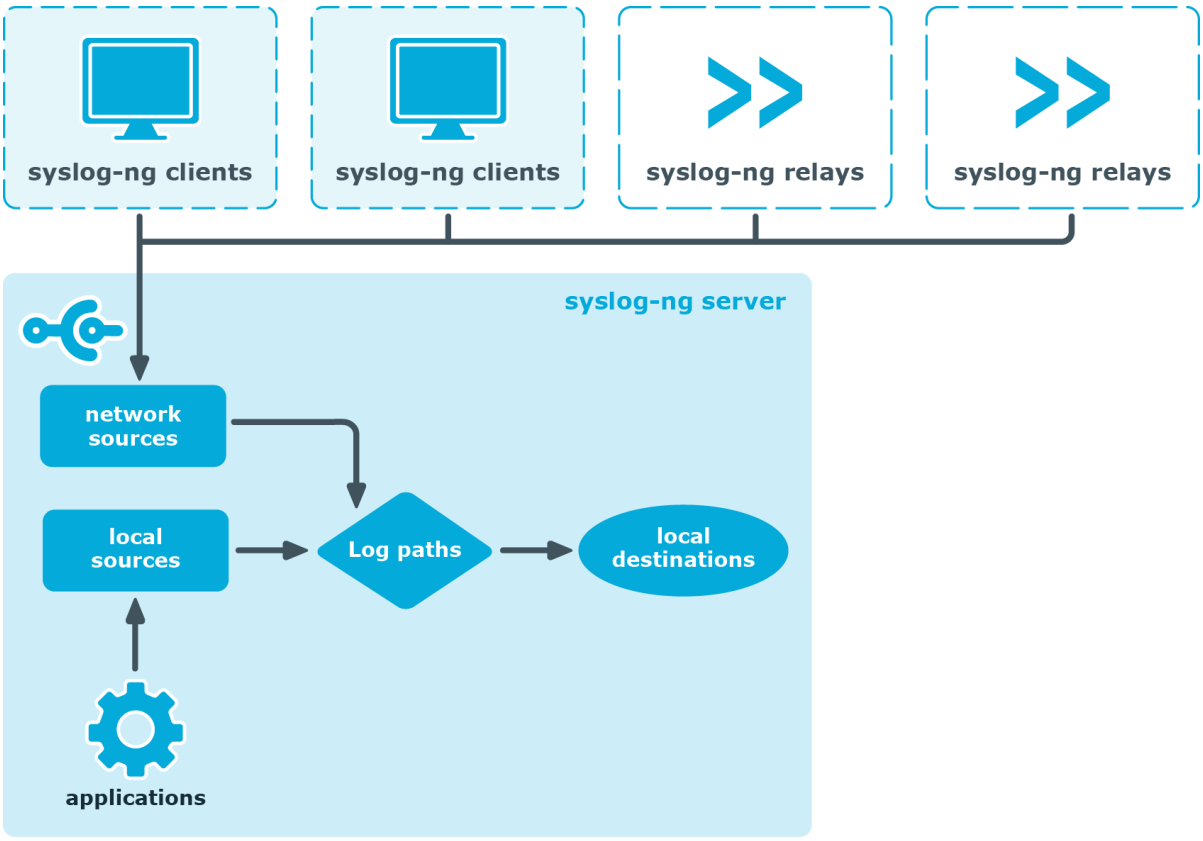
What syslog-ng relays are not good for

The purpose of the relay is to buffer the logs for short term, for example, a few minutes or a few hours long outages (depending on the log volume). It is not designed to buffer logs generated by the sources during a very long server or connection outage, for example, up to a few days long.

If you expect extended outages, One Identity recommends that you deploy servers instead of relays. There are many deployments where long term storage and archiving are performed on the central syslog-ng server, but relays also do short-term log storage. From the syslog-ng PE point of view, these are servers, and thus need separate server licenses.

Server mode

Figure 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 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 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 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 Customizing message format using macros and templates.
- *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 Customizing message format using macros and templates.
- *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 parser: Parse and segment structured messages and 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 Modifying messages using rewrite rules.
- *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 Routing messages: log paths, flags, 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 Customizing message format using macros and templates.
- *Option*: Options set global parameters of syslog-ng, like the parameters of name resolution and timezone handling. For details, see Global options of syslog-ng PE.

For details on the above objects, see Global objects.

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.

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 (winter-time) 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 Premium Edition

syslog-ng PE

The following release policy applies to syslog-ng Premium Edition (syslog-ng PE):

Long Term Support (LTS)

The initial release includes new features, bug fixes and security updates. After the initial release, only maintenance releases are published on this path, containing only bug fixes and security updates. The maintenance release frequency is typically four months.

Versioning: the first digit identifies the LTS main version (for example, 6.0.x), the second digit is always a 0, and the third digit designates the maintenance release (for example, 6.0.19). A long term support path is typically supported for three years after its original release.

Rolling release

Rolling releases include new features, bug fixes and security updates. Release frequency on this path is typically two months.

Versioning: the first digit identifies the main version of the rolling release path, the second digit is always a 0, and the third digit designates published on this path. Rolling releases are typically supported for a year.

For further information regarding the syslog-ng PE LTS and Rolling releases, see the syslog-ng Premium Edition Product Life Cycle Table.

CAUTION:

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

Logstore files created with a newer version of syslog-ng PE might not be readable with an older version of syslog-ng PE.

NOTE: Bug fixes and security updates are always issued in the latest & greatest releases, and never for previous releases. For example, in case of Long Term Support path, if a bug was reported by a customer for 6.0.17 LTS, the fix will be released in version 6.0.18 or in a later maintenance release. The same logic is true to rolling releases, for example, if a bug gets reported for 7.0.20, the fix will be issued in 7.0.21 or a later release.

NOTE: The LTS path for syslog-ng PE will contain support only for the Windows Agent and AIX components after 31-Jul-2020. All other platforms will be deprecated from the LTS path. One Identity advises customers to migrate to version 7.0.x where possible to be eligible for full support going forward.

Licensing

License 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 PE 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 syslog-ng.com.

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 Software Transaction, License and End User License Agreements and the 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 the disk-buffer option 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.

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

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.

- 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 sources 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, ALTP).

  To make syslog-ng PE forget old clients that do not exist anymore, enable the `reset-license-counter()` global option.
- 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.

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.

Third-party contributions 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 BSD-syslog or legacy-syslog messages
- The new standard described in RFC 5424 (also called the IETF-syslog protocol): see IETF-syslog messages
- The Enterprise-wide message model or EWMM allows you to deliver structured messages between syslog-ng nodes: see Enterprise-wide message model (EWMM)
- How messages are represented in syslog-ng PE: see 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 Premium Edition (syslog-ng PE) application supports longer messages as well. For details, see the `log-msg-size()` option in Global options. However, it is not recommended to enable messages larger than the packet size when using UDP destinations.

The PRI message part

This section describes the PRI message part of a syslog message, according to the legacy-syslog or BSD-syslog protocol.

For further details about the HEADERand MSG parts of a syslog message, see the following sections:

- HEADER
- MSG

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.

PRI formula

The Priority value is calculated using the following formula:

```
<PRI> = ( <facility> * 8) + <severity>
```

That is, you first multiply the Facility number by 8, and then add the numerical value of the Severity to the multiplied sum.

Example: the correlation between Facility value, Severity value, and the Priority value in the PRI message part

The following example illustrates a sample syslog message with a sample PRI field (that is, Priority value):

```
<133> Feb 25 14:09:07 webserver syslogd: restart
```

In this example, <133> represents the PRI field (Priority value). The syslog message's Facility value is 16, and the Severity value is 5.

Substituting the numerical values into the <PRI> = (<facility> * 8) + <severity> formula, the results match the Priority value in our example:

<133> = (<16> * 8) + <5>.

Facility and Severity values

The possible Facility values (between 0 and 23) and Severity values (between 0 and 7) each correspond to a message type (see Table 1: syslog Message Facilities), or a message importance level (see Table 2: syslog Message Severities).

NOTE: Facility codes may slightly vary between different platforms. The syslog-ng Premium Edition (syslog-ng PE) application accepts Facility codes as numerical values as well.

The following table lists possible Facility values.

Table 1: 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 possible Severity values.

Table 2: 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

Numerical Code	Severity
6	Informational: informational messages
7	Debug: debug-level messages

The HEADER message part

This section describes the HEADER message part of a syslog message, according to the legacy-syslog or BSD-syslog protocol.

For further details about the PRI and MSG parts of a syslog message, see the following sections:

- PRI
- MSG

The HEADER message part

The HEADER message 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 Premium Edition (syslog-ng PE) application supports other timestamp formats as well, like ISO, or the PIX extended format. For details, see the ts-format() option in Global options.

The MSG message part

This section describes the MSG message part of a syslog message, according to the legacy-syslog or BSD-syslog protocol.

For further details about the HEADER and PRI message parts of a syslog message, see the following sections:

- HEADER
- PRI

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 (source: https://tools.ietf.org/html/rfc5424):

```
<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
```

- Facility is 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.
- In this example, the encoding is defined by the BOM:

  The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.
- There is no STRUCTURED-DATA present in the message, this is indicated by "-" in the STRUCTURED-DATA field.

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 3: 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 4: 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 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 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 theBOM[1]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.

Enterprise-wide message model (EWMM)

The following section describes the structure of log messages using the Enterprise-wide message model or EWMM message format.

The Enterprise-wide message model or EWMM allows you to deliver structured messages from the initial receiving syslog-ng component right up to the central log server, through any number of hops. It does not matter if you parse the messages on the client, on a relay, or on the central server, their structured results will be available where you store the messages. Optionally, you can also forward the original raw message as the first syslog-ng component in your infrastructure has received it, which is important if you want to forward a message for example, to a SIEM system. To make use of the enterprise-wide message model, you have to use the syslog-ng() destination on the sender side, and the default-network-drivers() source on the receiver side.

The following is a sample log message in EWMM format.

```
<13>1 2018-05-13T13:27:50.993+00:00 my-host @syslog-ng - - -
{"MESSAGE":"<34>Oct 11 22:14:15 mymachine su: 'su root' failed for username on
/dev/pts/8","HOST_FROM":"my-host","HOST":"my-host","FILE_NAME":"/tmp/in","._
TAGS":".source.s_file"}
```

The message has the following parts:

- The header of the complies with the RFC5424 message format, where the PROGRAM field is set to `@syslog-ng`, and the SDATA field is empty.
- The MESSAGE part is in JSON format, and contains the actual message, as well as any name-value pairs that syslog-ng PE has attached to or extracted from the

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

message. The `${._TAGS}` field contains the identifier of the syslog-ng source that has originally received the message on the first syslog-ng node.

To send a message in EWMM format, you can use the syslog-ng() destination driver, or the format-ewmm() template function.

To receive a message in EWMM format, you can use the default-destination-drivers() source driver, or the ewmm-parser() parser.

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 parser: Parse and segment 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 Hard versus 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 3.35.1 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:

- `amqp()` destination
- `format-welf()` template function
- `mongodb()` destination
- `stomp()` 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 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, AMQP) 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 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: 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.
- `double`: A floating-point number.
- `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: 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. `scope()`
2. `exclude()`
3. `key()`
4. `pair()`

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")
)
```

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

omit-empty-values()

Type:	flag
Default:	N/A

Description: If this option is specified, syslog-ng PE does not include value-pairs with empty values in the output. For example: `$(format-json --scope none --omit-empty-values)` or

```
value-pairs(
    scope(rfc3164 selected-macros)
    omit-empty-values()
)
```

Available in syslog-ng PE version 7.0.143.21 and later.

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:

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 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:
Table Caption Outside Table: 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`, `$R_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 also has the following alias: `syslog-proto`. Note that the value of `$R_DATE` will be listed under the `DATE` key.

  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)
)
```

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 ALTP (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. `syslog(transport(tcp))`
2. `syslog(transport(udp))`
3. `syslog(transport(altp))`
4. `syslog(transport(tls))`
5. `syslog(transport(altp(tls-required(yes)))`
6. `network(transport(tcp))`
7. `network(transport(udp))`
8. `network(transport(altp))`
9. `network(transport(tls))`
10. `network(transport(altp(tls-required(yes)))`
11. `network(transport(tcp) flag(syslog-protocol))`
12. `network(transport(udp) flag(syslog-protocol))`
13. `network(transport(altp)flag(syslog-protocol))`
14. `network(transport(tls) flag(syslog-protocol))`
15. `network(transport(altp(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 6: Source-destination driver combinations

Source \ Destination	syslog/tcp	syslog/udp	syslog/tls	network/tcp	network/udp	network/tls	network/tcp/-flag	network-/udp/-flag	network/tls/-flag
syslog/tcp	✓	-	-	!	-	-	!	-	-
syslog/udp	-	✓	-	-	!	-	-	!	-
syslog/tls	-	-	✓	-	-	!	-	-	!
network/tcp	-	-	-	✓	-	-	✓ ?	-	-
network-	-	✓ ?	-	-	✓	-	-	✓ ?	-

Source \ Destination	syslog/tcp	syslog/udp	syslog/tls	network/tcp	network/udp	network/tls	network/tcp/-flag	network/udp/-flag	network/tls/-flag
k/udp									
network/tls	-	-	-	-	-	✓	-	-	✓ ?
network/tcp/-flag	!	-	-	!	-	-	✓	-	-
network/udp/-flag	-	!	-	-	!	-	-	✓	-
network/tls/-flag	-	-	!	-	-	!	-	-	✓

- - 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.

NOTE: To receive UDP messages at very high message rate, you can use the `udp-balancer()` source. For details, see udp-balancer: Receiving UDP messages at very high rate.

Using syslog-ng PE with NFS or CIFS (or SMB) file system for log files

This section and its subsections describe using syslog-ng Premium Edition (syslog-ng PE) with NFS or CIFS file system for log files.

NOTE: All information referring to the CIFS network file system in this section automatically applies to the SMB network file system as well. As a result, even if the content of this section only refers to the CIFS file system, keep in mind that the same information also applies to SMB file systems as well.

If your NFS or CIFS connection is not stable, using the NFS or CIFS network file system can lead to problems. As a result, One Identity does neither recommend, nor officially support such scenarios. If you can avoid it, do not store log files on NFS or CIFS. If the NFS or CIFS connection is stable and reliable, syslog-ng Premium Edition (syslog-ng PE) can read and

write files on mounted NFS or CIFS partitions as a normal file source or destination. Read this section carefully before using syslog-ng PE and NFS-mounted or CIFS-mounted log files.

Limitations of using syslog-ng PE with NFS or CIFS (or SMB) file system

This section lists the limitations of using syslog-ng PE with NFS or CIFS file system.

NOTE: All information referring to the CIFS network file system in this section automatically applies to the SMB network file system as well. As a result, even if the content of this section only refers to the CIFS file system, keep in mind that the same information also applies to SMB file systems as well.

Using syslog-ng PE with NFS or CIFS file system has the following limitations:

- **CAUTION:**

  Hazard of data loss!

  The syslog-ng PE application does not support storing the disk-buffer files on any kind of network share (that is, NFS, CIFS, and so on). To avoid crashes, data corruption, or data loss, One Identity does not recommend storing your disk-buffer files on network share.

- Do not use the `logstore()` destination to store files on an NFS-mounted or CIFS-mounted partition.
- To use wildcards in the file source, set the `force-directory_polling()` option to `yes` to detect newly created files. Note that this option is available only in syslog-ng PE version 6.0.3 and newer versions of the 6.x branch, and is 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 or CIFS, One Identity will handle support requests and bugs related to such scenarios only if you can reproduce the issue independently from NFS or CIFS.

Risks of using syslog-ng PE with NFS or CIFS (or SMB) file system

This section describes the risks of using syslog-ng PE with NFS or CIFS file system.

NOTE: All information referring to the CIFS network file system in this section automatically applies to the SMB network file system as well. As a result, even if the content of this section only refers to the CIFS file system, keep in mind that the same information also applies to SMB file systems as well.

CAUTION:

Hazard of data loss!

The syslog-ng PE application does not support storing the disk-buffer files on any kind of network share (that is, NFS, CIFS, and so on). To avoid crashes, data corruption, or data loss, One Identity does not recommend storing your disk-buffer files on network share.

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

- The syslog-ng PE application 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.
- The syslog-ng PE application cannot 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.

Recommendations for using syslog-ng PE with NFS or CIFS (or SMB) file system

This section lists the recommendations for using syslog-ng Premium Edition (syslog-ng PE) with NFS or CIFS file system.

NOTE: All information referring to the CIFS network file system in this section automatically applies to the SMB network file system as well. As a result, even if the content of this section only refers to the CIFS file system, keep in mind that the same information also applies to SMB file systems as well.

CAUTION:

When you use NFS with syslog-ng PE, One Identity recommends the following:

- **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 a logstore on an NFS-mounted partition. It can easily become corrupted.**

CAUTION:

When you use CIFS with syslog-ng PE, One Identity recommends the following:

- **DO NOT install syslog-ng PE on a CIFS-mounted partition.**
- **DO NOT store the runtime files (for example, the configuration or the persist file) of syslog-ng PE on a CIFS-mounted partition.**
- **DO NOT use a logstore on a CIFS-mounted partition. It can easily become corrupted.**

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.

4

Installing syslog-ng PE

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 platform-specific installers, for example, RPM, is described in the following sections.

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 reuse 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 Installing syslog-ng using the .run installer, and also without user interaction using the silent installation option — see Installing syslog-ng PE without user-interaction.

NOTE: When setting up a virtual environment, carefully consider the configuration aspects such as CPU, memory availability, I/O subsystem, and network infrastructure to ensure the virtual layer has the necessary resources available. Please consult One Identity's Product Support Policies for more information on environment virtualization.

Prerequisites to installing syslog-ng PE

- The binary installer packages of syslog-ng Premium Edition (syslog-ng PE) 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.
-

Supported OpenSSL versions

The following list contains information about the supported OpenSSL versions in each syslog-ng PE application version.

Linux glibc 2.11

syslog-ng PE version	supported Open SSL version
7.0.1	OpenSSL 1.0.2j
7.0.2	OpenSSL 1.0.2j
7.0.3	OpenSSL 1.0.2j
7.0.4	OpenSSL 1.0.2j
7.0.5	OpenSSL 1.0.2j
7.0.6	OpenSSL 1.0.2m
7.0.7	OpenSSL 1.0.2m
7.0.8	OpenSSL 1.0.2m
7.0.9	OpenSSL 1.0.2o
7.0.10	OpenSSL 1.0.2o
7.0.11	OpenSSL 1.0.2p
7.0.12	OpenSSL 1.0.2q
7.0.13	OpenSSL 1.0.2q
7.0.14	OpenSSL 1.0.2r
7.0.15	OpenSSL 1.0.2s
7.0.16	OpenSSL 1.0.2s
7.0.17	OpenSSL 1.0.2t
7.0.18	OpenSSL 1.0.2t

syslog-ng PE version	supported Open SSL version
7.0.19	OpenSSL 1.1.1d
7.0.20	OpenSSL 1.1.1g
7.0.21	OpenSSL 1.1.1g
7.0.22	OpenSSL 1.1.1g
7.0.23	OpenSSL 1.1.1h
7.0.24	OpenSSL 1.1.1j
7.0.25.	OpenSSL 1.1.1k
7.0.26	OpenSSL 1.1.1k
7.0.27	OpenSSL 1.1.1l

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 Using syslog-ng PE on SELinux.

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

The following describes how to install syslog-ng PE on operating systems that use the Red Hat Package Manager (RPM). Installing syslog-ng PE automatically replaces the original syslog service. The following supported operating systems use RPM:

- Red Hat Enterprise Linux
- Red Hat Enterprise Server
- SUSE Linux Enterprise Server

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.

To install syslog-ng PE on operating systems that use the RPM

1. Login to the Support Portal and download the syslog-ng RPM package for your system.
2. - If the host already uses syslog-ng PE for logging, execute the following command as root. Otherwise, skip this step.

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

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

     NOTE: If you are upgrading from syslog-ng version 2.1, note that the location of the configuration file has been moved to `/opt/syslog-ng/etc/syslog-ng.conf`
   - Execute the following command as root:

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

     The syslog-ng PE application and all its dependencies will be installed.
3. **CAUTION:**

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

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

   This behavior has been detected on CentOS 4 systems, but may occur on other rpm-based platforms as well.
4. Edit the syslog-ng PE configuration file as needed. If you want to run syslog-ng PE in server mode, copy the license file to the `/opt/syslog-ng/etc/` directory.

   For information on configuring syslog-ng PE, see the The syslog-ng PE quick-start guide.

5. (Optional step for SELinux-enabled systems): Complete Using syslog-ng PE on SELinux.

Using syslog-ng PE on SELinux

Version syslog-ng PE 5 F2 and later properly supports SELinux on Red Hat Enterprise Linux 6.5 and newer platforms. Version 5 F5 and later also supports SELinux on Red Hat Enterprise Linux 5, as well as on 6.0-6.4. The CentOS and Oracle Linux platforms corresponding to the supported RHEL versions are supported as well. To use syslog-ng PE on a SELinux-enabled host, complete the following steps:

NOTE: The following steps install SELinux policy module that enables syslog-ng PE to properly run with its default configuration and default installation path (`/opt/syslog-ng`) on a SELinux-enabled host. If you configure syslog-ng PE to perform an operation that is outside the permissions of this policy module (for example, to bind to a non-standard port, use a program destination or source, or to write logfiles in a non-standard directory), you have to modify and recompile the policy module. If you need help with that, access the Support Portal. For contact details, see About us.

Prerequisites

- The following packages must be available on the host: `policycoreutils`,`policycoreutils-devel`, `policycoreutils-python`. If they are not already installed, issue the following command: `yum install policycoreutils policycoreutils-devel policycoreutils-python`
- On RHEL 6.5, update the following packages at least to the indicated versions. These packages are available in the Red Hat repositories and are installed by default on RHEL 6.6. You can update them with the `yum update selinux-policy` command:
  - `selinux-policy-3.7.19-231.el6.noarch > 3.7.19-260.el6.noarch`
  - `selinux-policy-targeted-3.7.19-231.el6.noarch > 3.7.19-260.el6.noarch`
- The syslog-ng PE application must be installed on the host. For details, see Installing syslog-ng PE.

Installing the SELinux Policy Module

The SELinux Policy Module is included in the syslog-ng PE RHEL package.

NOTE: Only the unmodified SELinux configuration is supported.

To install the SELinux Policy Module,

1. Run the `/opt/syslog-ng/share/doc/selinux/syslog_ng.sh` script to compile and load the SELinux rules for syslog-ng PE.
2. Restart syslog-ng PE using the following command.

CAUTION:

The SELinux policy works only if syslog-ng PE is started by the init daemon.

- On RHEL6: `service syslog-ng restart`
- On RHEL7: `systemctl restart syslog-ng`

If you do not use the `service` or the `systemctl` to start syslog-ng PE, run the `syslog_ng.sh` script again after starting syslog-ng PE. This is required to correct the settings of the files related to syslog-ng PE (most notably `/dev/log` and the files under `/opt/syslog-ng`). The settings can become incorrect if the privileges of the process that started syslog-ng PE are different from the privileges of the `service` or the `systemctl` process.

3. (Optional): The syslog-ng PE application can create coredumps, but this is disabled by default. You can enable coredumps with the `setsebool -P daemons_dump_core 1` command.

   Note that his command enables every daemons on your system to create core dumps, not just syslog-ng PE. There is no way to enable per-application core dumps in SELinux.

Expected result

The syslog-ng PE application is installed and properly running under SELinux. If syslog-ng PE does not start, or displays permission errors, run the `syslog_ng.sh` script.

Installing syslog-ng PE on Debian-based platforms

The following describes how to install syslog-ng Premium Edition (syslog-ng PE) on operating systems that use the Debian Software Package (deb) format. The following supported operating systems use this format:

- Ubuntu 16.04 LTS (Xenial Xerus)
- Ubuntu 18.04 LTS (Bionic Beaver)

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.

To install syslog-ng PE on operating systems that use the Debian Software Package (deb) format

1. Login to your syslog-ng PE account and download the syslog-ng PE DEB package for your system.
2. Issue the following command as root:

   ```
   dpkg -i syslog-ng-premium-edition-<version>-<OS>-<arch>.deb
   ```

3. Answer the configuration questions of syslog-ng PE. These are described in detail in Installing syslog-ng using the .run installer.

   For information on configuring syslog-ng PE, see the The syslog-ng PE quick-start guide.

Installing syslog-ng in Docker

The following describes how to install syslog-ng PE in a Docker container. The following operating systems are supported:

- CentOS 7
- RedHat EL 7.5
- Ubuntu 18.04 LTS (Bionic Beaver)

To install syslog-ng PE in a Docker container

1. Start Docker. Use the command appropriate for you platform:
   - `docker run -d -p <network-ports-forwarded-to-docker> -v <directories-to-be-mounted> --name syslog-ng-in-docker centos:7`
   - `docker run -d -p <network-ports-forwarded-to-docker> -v <directories-to-be-mounted> --name syslog-ng-in-docker registry.access.redhat.com/rhel-7.5-s390x`
   - `docker run -d -p <network-ports-forwarded-to-docker> -v <directories-to-be-mounted> --name syslog-ng-in-docker ubuntu:18.04`

   For example, to forward port 514 and mount the etc and var directories on RedHat, use the following command: `docker run -d -p 514:514 -v /root/docker/etc/:/opt/syslog-ng/etc -v /root/docker/var:/opt/syslog-ng/var --name syslog-ng-in-docker registry.access.redhat.com/rhel-7.5-s390x`

   Note the following points:
   - Forward all ports to Docker that you want to receive messages from in your syslog-ng PE configuration.
   - The previous example mounts the etc and var directories from outside the docker container. That way you can edit the syslog-ng PE configuration file outside the container, and the syslog-ng PE persist file will not be deleted if you delete and recreate the docker container.

- Do not mount the same var directory for multiple docker containers.
- Make sure that the syslog-ng PE running in the docker container has permissions to read the configuration file, and read and write permissions for the var directory.
- If you want to read the logs of the host from `/dev/log`, mount it into the Docker container. Note that only a single syslog-ng PE instance can read `/dev/log` at the same time. Do not mount the same `/dev/log` for multiple syslog-ng PE instances.

2. Download the syslog-ng PE .run installation package from Downloads page.
3. Install syslog-ng PE in the Docker container. (Since there is no service management (systemd) in the docker container, the registration and start of the syslog-ng PE service is disabled.)

   ```
   docker exec -it syslog-ng-in-docker /bin/bash
   ```

   ```
   syslog-ng-premium-edition-7.0.29-linux-glibc2.11-amd64.run -- --accept-eula --silent --no-register
   ```
4. (Optional Step) If you want to use any features of syslog-ng PE that require external packages (for example, Java or Python-based destinations), install the required packages manually in the Docker container (for example, Java or Python).
5. Start syslog-ng PE.

   ```
   docker exec -i syslog-ng-in-docker /opt/syslog-ng/sbin/syslog-ng <-optional-command-line-parameters-of-syslog-ng>
   ```

   For the list of available command-line parameters, see the syslog-ng.8 manual page.

Start, reload, stop syslog-ng PE in a Docker container

To start syslog-ng PE, issue the following command in the Docker container.

```
docker exec -i syslog-ng-in-docker /opt/syslog-ng/sbin/syslog-ng <-optional-command-line-parameters-of-syslog-ng>
```

To reload syslog-ng PE, issue the following command in the Docker container.

```
docker exec -i syslog-ng-in-docker /opt/syslog-ng/sbin/syslog-ng-ctl reload
```

To stop syslog-ng PE, issue the following command in the Docker container.

```
docker exec -i syslog-ng-in-docker /opt/syslog-ng/sbin/syslog-ng-ctl stop
```

Upgrading syslog-ng PE running in a Docker container

To upgrade a syslog-ng PE instance that is running in a Docker container

1. Download the new syslog-ng PE .run installation package from Downloads page.
2. Upgrade syslog-ng PE in the Docker container.

   ```
   docker exec -it syslog-ng-in-docker /bin/bash
   ```

```
syslog-ng-premium-edition-7.0.29-linux-glibc2.11-amd64.run -- --accept-eula --silent --no-register --upgrade
```

3. Start syslog-ng PE.

   ```
   docker exec -i syslog-ng-in-docker /opt/syslog-ng/sbin/syslog-ng <-optional-command-line-parameters-of-syslog-ng>
   ```

   For the list of available command-line parameters, see the syslog-ng.8 manual page.

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 Installing syslog-ng PE in client or relay mode.
- To install syslog-ng PE on your central log server, complete Installing syslog-ng PE in server mode.
- To install syslog-ng PE without any user-interaction, complete 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.

Installing syslog-ng PE in client or relay mode

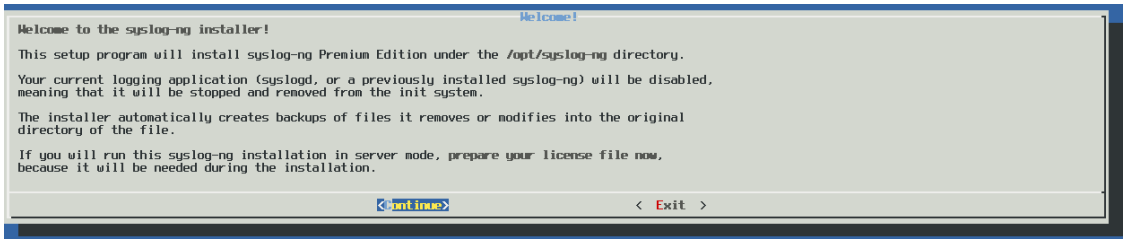
The following describes how to install syslog-ng Premium Edition on clients or relays. For details on the different operation modes of syslog-ng PE, see Modes of operation.

To install syslog-ng Premium Edition on clients or relays

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 the Support Portal and download the syslog-ng PE installer package.
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**.

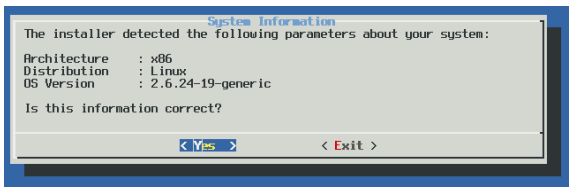
Figure 5: The welcome screen



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 in this guide for convenience at Software Transaction, License and End User License Agreements. Select **Accept** to accept the EULA and continue the installation.

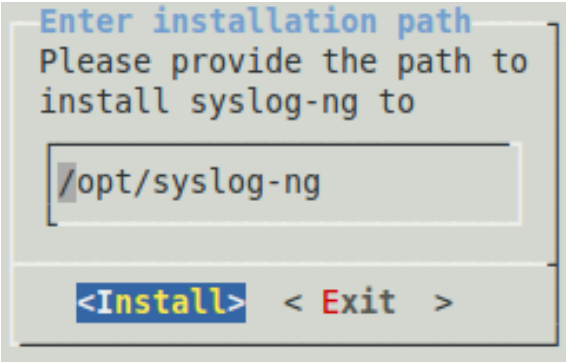
   If you do not accept the terms of the EULA for some reason, select **Reject** to cancel installing syslog-ng PE.
4. *Detecting platform and operating system*: The installer attempts to automatically detect your oprating system and platform. If the displayed information is correct, select **Yes**. Otherwise select **Exit** to abort the installation, and verify that your platform is supported. For a list of supported platforms, see Supported platforms. If your platform is supported but not detected correctly, contact your local distributor, reseller, or access the Support Portal. For contact details, see About us.

Figure 6: Platform detection



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.

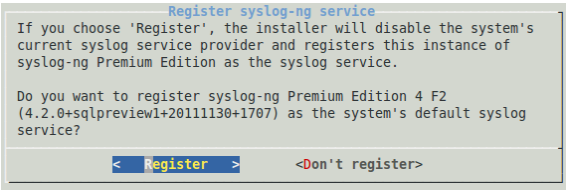
Figure 7: Installation path



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.

6. *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.

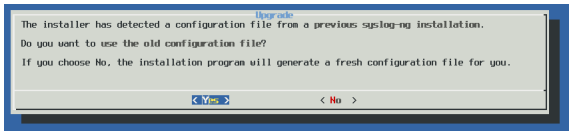
Figure 8: Registering as syslog service



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 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 9: Upgrading syslog-ng



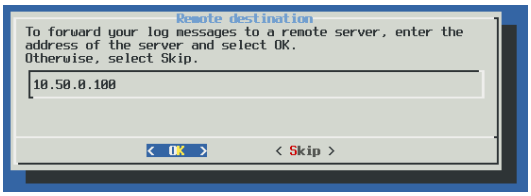
9. *Generating a new configuration file*: The installer displays some questions to generate a new configuration file.
    a. *Remote sources*: Select **Yes** to accept log messages from the network. TCP, UDP, and SYSLOG messages on every interface will be automatically accepted.

    Figure 10: Accepting remote messages

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

    Figure 11: 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 Using syslog-ng PE on SELinux.

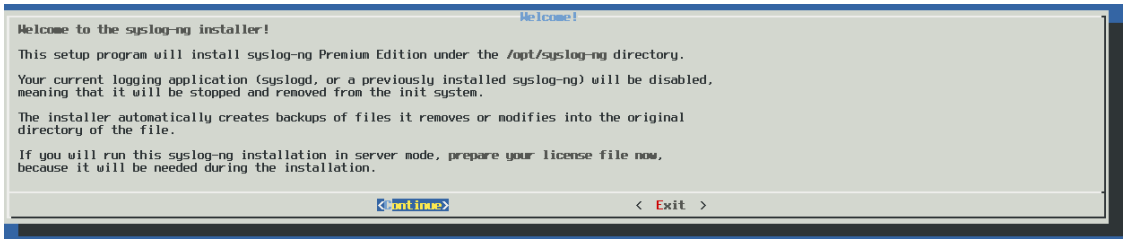
Installing syslog-ng PE in server mode

The following describes how to install syslog-ng PE on log servers. For details on the different operation modes of syslog-ng PE, see Modes of operation.

To install syslog-ng PE on log servers

1. Login to the Support Portal 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 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**.

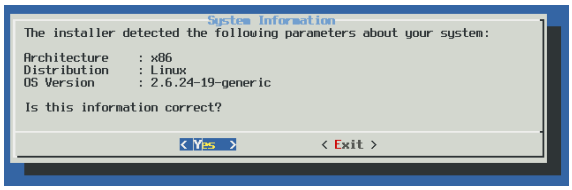
   Figure 12: The welcome screen

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 in this guide for convenience 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.
4. *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 Supported platforms. If your platform is supported but not detected correctly, contact your local distributor, reseller, or access the Support Portal. For contact details, see About us.

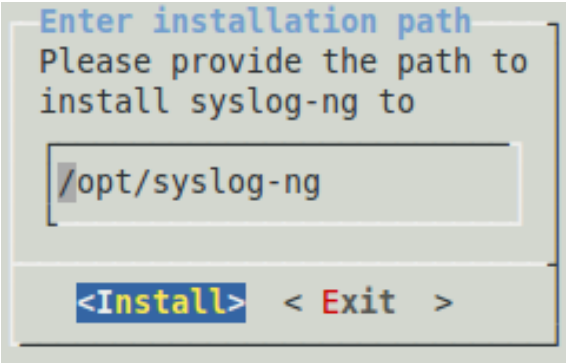
   Figure 13: Platform detection

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.

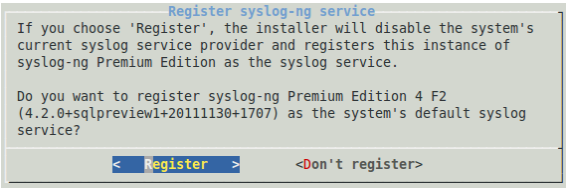
Figure 14: Installation path



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.

6. *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.

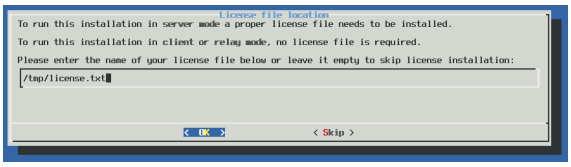
Figure 15: Registering as syslog service



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.

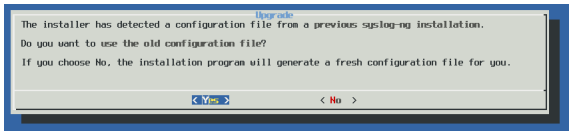
Figure 16: Platform detection



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 17: Upgrading syslog-ng



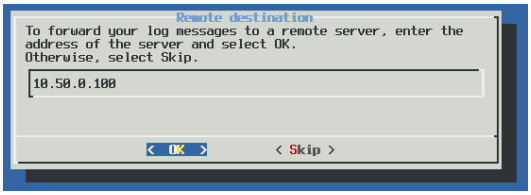
9. *Generating a new configuration file*: The installer displays some questions to generate a new configuration file.
    a. *Remote sources*: Select **Yes** to accept log messages from the network. TCP, UDP, and SYSLOG messages on every interface will be automatically accepted.

    Figure 18: Accepting remote messages

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

    Figure 19: 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
```

    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.

11. (Optional step for SELinux-enabled systems): Complete 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 directly to syslog-ng PE version 7.0.29 is supported from syslog-ng PE version 7.0.x. To upgrade from an older 7.0.x version to version 7.0.29, see Upgrading from syslog-ng PE 7.0.x to version 7.
- To upgrade a syslog-ng PE 6 LTS (6.0.x) installation to syslog-ng PE version 7.0.x, see Upgrading from syslog-ng PE 6.0.x to version 7.

Upgrading from syslog-ng PE 7.0.x to version 7

The following describes how to upgrade to syslog-ng PE 7.

Prerequisites

- You must have a valid software subscription to be able to download the new version of syslog-ng PE.

To upgrade fromsyslog-ng PE 7.0.x to 7 on a production machine

1. Download the installer package for version 7 of syslog-ng PE from the Downloads page. Use the same package type as you used for the installation (for example, use the `.run` package for the upgrade if you have originally installed syslog-ng PE using a `.run` installer)
2. Login to the host you want to upgrade.
3. Install syslog-ng PE and check any warnings. Upgrade the respective parts of your configuration if needed.
4. Restart syslog-ng PE and verify that it is running and processing logs properly. For details, see Managing and checking syslog-ng service on Linux.
5. If you experience problems during the upgrade, contact our Support Team.

Upgrading from syslog-ng PE 6.0.x to version 7

The following describes how to upgrade to syslog-ng PE 7.

Notes and warnings about the upgrade

CAUTION:

Read the entire document thoroughly before starting the upgrade.

CAUTION:

Hazard of data loss!

Consider the following before upgrading from syslog-ng PE 6.0.x to version 7:

- **When upgrading a syslog-ng PE installation that is running in server or relay mode, One Identity recommends redirecting the client logs to another syslog server to avoid message loss.**
- **During the upgrade, there will be periods when syslog-ng PE will not be able to receive logs. Make sure to upgrade in a dedicated maintenance window when you can disable receiving logs.**

- Versions 6.0.x and 7.0.x are significantly different, therefore direct upgrade is not possible. Read the entire document thoroughly before starting the upgrade. For a detailed list about the differences, see "Differences in features between syslog-ng PE 6 LTS and 7" in the Release Notes.
- The list of supported platforms has changed in version 7.0.x. Verify that your platform is supported on the Supported platforms page. In particular, 32-bit systems are no longer supported, so they can not be upgraded to version 7.0.x.
- Version 7.0.x of syslog-ng PE does not support every feature of 6.0.x. For a detailed list about the differences, see "Differences in features between syslog-ng PE 6 LTS and 7" in the Release Notes.
- Features available in both 7.0.x and 6 LTS may work differently. For a detailed list about the differences, see "Differences in features between syslog-ng PE 6 LTS and 7" in the Release Notes.

Prerequisites

- You must have a valid software subscription to be able to download the new version of syslog-ng PE .

Complete the following steps before starting the actual upgrade process.

1. Download the installer package for the latest 7.0.x version of syslog-ng PE from the Downloads page.
2. If you are using syslog-ng PE in server mode, download the new license from My License Assets.
3. Install syslog-ng PE 7 on a test machine.
4. Replace the syslog-ng PE configuration file on the test machine with your syslog-ng PE version 6.0.x configuration file.
5. Edit the configuration file: update the version string to `@version: 7.0`.
6. Update the configuration file.

   Some features and functionalities have changed between version 6 and 7 and require you to change the configuration file, for example, to delete options that have been removed from the product, or change the names of options that have been renamed.

For a detailed list about the differences, see "Differences in features between syslog-ng PE 6 LTS and 7" in the Release Notes.

Perform the following steps until you have an updated configuration file that properly works with syslog-ng PE 7.

a. Check the configuration for syntax errors using the `syslog-ng --syntax-only --cfgfile <your-modified-configuration-file>` command. Modify your configuration to correct any errors and warnings.

b. Restart syslog-ng.

c. Verify that message filtering and processing (for example, parsing and rewriting) works properly.

d. Verify that the format and content of the output messages is correct, including the SDATA of RFC-5424-formatted messages if you are using them.

To upgrade from syslog-ng PE 6.0.x to 7 on a production machine

1. If you use disk buffers, empty your disk-buffer files before starting the upgrade process.

   For more information about disk-buffer files and how to get information about them, see How to get information about disk-buffer files.

   For more information about emptying your disk-buffer files, see How to empty disk-buffer files.

   CAUTION:

   Hazard of data loss!

   If you have logs in your disk-buffer files, failing to empty the disk-buffer files before starting the upgrade process may result in log loss. One Identity recommends that you empty your disk-buffer files before starting the upgrade process.

2. Before upgrading the host to syslog-ng PE 7, upgrade it to the latest available 6.0.x version.

3. Login to the host you want to upgrade.

   CAUTION:

   Hazard of data loss!

   This host will not be able to receive or send log messages until the upgrade procedure is complete. Make sure to upgrade in a dedicated maintenance window when you can disable receiving logs.

   When upgrading a syslog-ng PE installation that is running in server or relay mode, One Identity recommends redirecting the client logs to another syslog server to avoid message loss.

4. Stop syslog-ng PE version 6.

5. Verify that syslog-ng PE has stopped. The output of the following command must be empty: `ps axu | grep syslog-ng`
6. Create a backup of the syslog-ng PE installation directory using the following command: `cp -a /opt/syslog-ng /opt/syslog-ng-pe6-backup`
7. Uninstall syslog-ng PE version 6. For details, see Uninstalling syslog-ng PE 6.
8. Delete the installation directory using the following command: `rm -rf /opt/syslog-ng`
9. Download the new installer package from the Downloads page.
10. Install syslog-ng PE 7. For details, see "Installing syslog-ng PE" in the Administration Guide.

    NOTE: If you want to run syslog-ng PE under a non-privileged user, see this knowledgebase article for details.
11. Stop syslog-ng PE version 7 started by the installer.
12. Copy the syslog-ng PE version 7 configuration file that you created on your test machine into the `/opt/syslog-ng/etc` directory of the host you are upgrading.
13. If the host you are upgrading is running syslog-ng PE in server mode, replace the old (version 6) license file with the new one.

    CAUTION:

    Hazard of data loss! Without the new license file, syslog-ng PE will run in relay mode, and will not store the incoming messages locally.
14. Restart syslog-ng PE and verify that it is running and processing logs properly. For details, see Managing and checking syslog-ng service on Linux.
15. If you experience problems during the upgrade, contact our Support Team.

Upgrading syslog-ng PE to other package versions

This scenario is not supported and will fail with the following error messages.

Upgrading from platform-specific package to .run

Upgrading from rpm package to .run package

Unsupported. Installation stops and the following error message is displayed:

```
Incompatible syslog-ng package already installed
```

Upgrading from deb package to .run package

Unsupported. Installation stops and the following error message is displayed:

```
Incompatible syslog-ng package already installed
```

Upgrading from pkg package to .run package

Unsupported. Installation stops and the following error message is displayed:

```
Incompatible syslog-ng package already installed to <syslog-ng path>
```

Upgrading from .run to a platform-specific package

This scenario is not supported and will fail with the following error messages. To replace a .run package with a platform-specific package, create a backup of your configuration and persist files, uninstall the .run package using the `--purge` option, then install the platform-specific package.

Upgrading from .run package to rpm package

Unsupported. Installation stops and the following error message is displayed:

```
Incompatible standalone (.run) installer of syslog-ng Premium Edition
```

CAUTION:

Hazard of data loss! Installing rpm package syslog-ng PE on AIX platform is possible even if the upgrade conditions are not met, since the rpm package installs before checking the upgrade conditions and therefore no error message is displayed. This might result in overwriting the old configuration file.

Upgrading from .run package to deb package

Unsupported. Installation stops and the following error message is displayed:

```
Errors were encountered while processing
```

Upgrading from .run package to pkg package

Unsupported. Installation stops and the following error message is displayed:

```
Please remove the conflicting package before installing this package.
Installation aborted.
```

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.

Upgrade from syslog-ng OSE to syslog-ng PE

If you wish to upgrade your existing syslog-ng OSE installation to syslog-ng PE, there are a number of considerations to keep in mind. This section highlights the main differences between syslog-ng OSE and syslog-ng PE that are useful to know before performing an upgrade. It also provides you with step-by-step instructions on how to do the upgrade.

Feature differences

syslog-ng OSE includes certain features that are highly experimental, require special external dependencies, or are important only to a very limited set of users. While syslog-ng PE is built from the same code base, it includes only a subset of syslog-ng OSE features. Those that are well tested and represent commercial value. These features are commercially supported as they are covered by automated end-to-end tests, which make sure that they not only compile but work correctly on many different platforms.

This means that your syslog-ng OSE installation may contain features that are not part of syslog-ng PE, or if they are, they may not have been tested.

Packaging

The packaging of syslog-ng OSE and syslog-ng PE also differ greatly.

With syslog-ng OSE, distribution packages do not bundle dependencies and only include features for which dependencies are available within the distribution. Packaging is modular to make sure that you install only a minimal set of extra dependencies. In addition, the naming and content of subpackages varies between distributions, and there are also unofficial syslog-ng OSE packages enabling more features than available in official distribution packages.

In the case of syslog-ng PE, all dependencies are included in a single package either in a distribution specific format (rpm or deb) or in a generic `.run` installer.

Upgrading from syslog-ng OSE to syslog-ng PE

The cleanest way to upgrade from syslog-ng OSE to syslog-ng PE is to remove the syslog-ng OSE package from the system. This way you can avoid the packaging conflicts and feature differences.

In the example procedure provided here, we describe an upgrade of syslog-ng OSE version 3.12 from unofficial repositories running on Red Hat Enterprise Linux 7.4 to syslog-ng PE version 7.0.4. The process should work in a fairly similar way when using other OS or syslog-ng versions.

To upgrade from syslog-ng OSE to syslog-ng PE

1. Remove syslog-ng OSE.

   The following instructions assume that the user is in the `/root` directory.

   a. Unless you have not touched the syslog-ng configuration at all, make a backup of `syslog-ng.conf` first. Copy the contents of `/etc/syslog-ng` to a directory under `/root` (or where you can find it), so you have a backup you can work from later:

   ```
   cp -R /etc/syslog-ng sngose
   ```

   b. Remove the syslog-ng package and dependent subpackages:

   ```
   yum erase syslog-ng
   ```

   c. Remove the `/etc/syslog-ng` directory:

   ```
   rm -fr /etc/syslog-ng
   ```

   > **CAUTION:**
   >
   > **Check the output of yum carefully. If there are any applications listed other than syslog-ng and subpackages, remove syslog-ng using `rpm -e –nodeps`, so dependent packages are not removed.**

2. Install syslog-ng PE.

   The following instructions assume that the syslog-ng PE rpm package is available in the current directory. You can install syslog-ng PE using the following command:

   ```
   [root@localhost ~]# rpm -Uvh syslog-ng-premium-edition-compact-7.0.5-
   1.rhel7.x86_64.rpm
   Preparing...                          ################################
   [100%]
   Trying to stop syslog services on Linux, using systemd services.
   Updating / installing...
      1:syslog-ng-premium-edition-compact################################
   [100%]
   Created symlink from /etc/systemd/system/multi-user.target.wants/syslog-
   ng.service to /usr/lib/systemd/system/syslog-ng.service.
   [root@localhost ~]#
   ```

3. Merge configurations.

   The configuration file of the freshly installed syslog-ng PE is available under `/opt/syslog-ng/etc/syslog-ng.conf`. Start by making a backup of it.

The next steps largely depend on the particulars of your previous syslog-ng OSE configuration and what you want to achieve:

a. Append your old OSE configuration to `/opt/syslog-ng/etc/syslog-ng.conf`.

b. Edit out redundant configuration parts, for example, a version declaration.

c. Edit out those configuration parts that refer to features unavailable in syslog-ng PE, such as the Riemann destination.

   If you try to start syslog-ng PE with an unknown feature enabled, it fails with a similar error message (in the example, it is the Riemann destination that is causing the error):

   ```
   /opt/syslog-ng/sbin/syslog-ng -s
   Error parsing destination, destination plugin riemann not found in
   /opt/syslog-ng/etc/syslog-ng.conf at line 41, column 2:

       riemann(
       ^^^^^^^
   ```

d. Syntax check your configuration using the `-s` option of syslog-ng. Make sure that you use the full path to syslog-ng PE, or add it to the PATH:

   ```
   /opt/syslog-nb/sbin/syslog-ng -s
   ```

e. If no errors are found, stop syslog-ng:

   ```
   systemctl stop syslog-ng
   ```

f. Try to start syslog-ng from the command line in the foreground using the `-F` option, so you can see any errors:

   ```
   /opt/syslog-ng/sbin/syslog-ng -F
   ```

   Some common error messages and explanations:

   - syslog-ng OSE uses `s_sys` for references to local system sources, while syslog-ng PE uses `s_local`. Remember to rename such references, otherwise a similar error message will be displayed:

     ```
     [2017-10-03T14:04:18.968550] Error resolving reference;
     content='source', name='s_sys', location='/opt/syslog-
     ng/etc/syslog-ng.conf:86:2'
     ```

   - Some features of syslog-ng PE require a license file to be present. In the example shown here, a Java plugin failed to initialize due to a missing license:

```
[2017-10-03T14:07:05.894534] syslog-ng running in client/relay
mode, cannot initialize plugin; plugin name='java'
[2017-10-03T14:07:05.894560] Error initializing message
pipeline; plugin name='java', location='#buffer:2:3'
```

Once you have made sure that your configuration works fine, you do not have to start syslog-ng in the foreground anymore.

g. Stop syslog-ng using `Ctrl-C`.

h. Start syslog-ng as a service using `systemctl start syslog-ng`.

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.
```

Upgrading the sql() source of syslog-ng PE

This section describes how you can upgrade your `sql()` source between syslog-ng Premium Edition (syslog-ng PE) 6 LTS and syslog-ng PE 7 LTS.

For more information about the `mssql()`, `oracle()`, or `sql()` sources, see mssql, oracle, sql: collecting messages from an SQL database.

To upgrade the `sql()` source between syslog-ng PE 6 LTS and syslog-ng PE 7 LTS

1. Extract the value of `last_read_uid` from the persist file by using persist tool.

   For more information on using the persist tool, see the The syslog-ng manual pages.

2. Add the extracted `last_read_uid` value to the syslog-ng PE configuration in a `start-uid(value)` format.

Example: adding the value of `last_read_uid` from the persist file into the configuration as `start-uid(value)`

You can add the value of `last_read_uid` that you extracted from the persist file into your configuration as `start-uid(value)`with the following method:

```
$ /opt/syslog-ng/bin/persist-tool dump /opt/syslog-ng/var/syslog-
ng.persist
hostid = { "value": "65 D5 E1 06" }

freetds,mssql-host,1433,test_database,test_table = { "version": 0, "big_
endian": false, "last_read_id": "2" }

run_id = { "value": "01 00 00 00" }
```

In the example above, the value is `"2"`. The parameter in the configuration should look like `start-uid("2")`.

Differences in configuration

This section describes the `sql()` source-related differences in configuration between syslog-ng PE 6 LTS and syslog-ng PE 7 LTS, including the features not ported from syslog-ng PE 6 LTS and the features that changed in usage since syslog-ng PE 6 LTS.

For more information about the `mssql()`, `oracle()`, or `sql()` sources, see mssql, oracle, sql: collecting messages from an SQL database.

Features not ported

- Only MSSQL and Oracle are supported in syslog-ng PE 7 LTS. MySQL and PostGreSQL users cannot upgrade to syslog-ng PE 7 LTS.
- `archive-query()`: It needs to be removed from configuration, because it is not supported in syslog-ng PE 7 LTS. Users need to find another way to cleanup old records.
- `read-old-records()`: This option is not supported in 7 LTS, as its functionality has been replaced by the `start-uid()` option. If you follow To upgrade the sql() source between syslog-ng PE 6 LTS and syslog-ng PE 7 LTS, your configuration will have a `start-uid()` in the appropriate configuration level and no specific action will be necessary.

Changes in features

- `default-facility()`

  The default value is `user` instead of `local0`.

  To achieve original behavior, you have to set `default-facility(local0)` explicitly in your configuration.

- `default-priority()`

  The default value is `notice` instead of `info`.

  In syslog-ng PE 7 LTS, `default-priority` has been renamed to `default-severity()`. For compatibility reasons, the original `default-priority()` option can be still used. However, One Identity recommends replacing it with `default-severity()` instead.

  To achieve original behavior, you have to set `default-severity(info)` explicitly in your configuration.

- `follow-freq()`

  The default value of `follow-freq()` has changed to `60` seconds (1 minute). The original default value was `10` seconds in syslog-ng PE 6 LTS.

  In syslog-ng PE 7 LTS, if `time-reopen()` is set, then syslog-ng PE will use the value you set. If neither `time-reopen()`, nor `follow-freq()` is set, the default value is `60` seconds.

  To achieve original behavior, you have to set `follow-freq(10)` explicitly in your configuration.

- `host-template()`

  In syslog-ng PE 6 LTS, it was mandatory to set `keep-hostname(yes)` for `host-template()` to work. In syslog-ng PE 7 LTS, syslog-ng PE will automatically set `keep-hostname(yes)` even if you do not set it in your configuration. If you set `host-template()`, but also set `keep-hostname(no)`, syslog-ng PE warns you about it, but otherwise ignores the `keep-hostname()` setting.

  In 6 LTS, syslog-ng PE allowed empty `host-template()` options in configuration. In syslog-ng PE 7 LTS, the template is mandatory in `host-template(template)`. The syslog-ng PE application will not start with such configuration. As a result, you have to delete empty `host-template()` options from your configuration.

- `message-template()`

  In 6 LTS, syslog-ng PE allowed empty `message-template()` options in configuration. In syslog-ng PE 7 LTS, the template is mandatory in `message-template(template)`. The syslog-ng PE application will not start with such configuration. As a result, you have to delete empty `message-template()` options from your configuration.

- `program-template()`

  In syslog-ng PE6 LTS, allowed empty `program-template()` options in configuration. In syslog-ng PE 7 LTS, the template is mandatory in `program-template(template)`. The syslog-ng PE application will not start with such configuration. As a result, you have to delete empty `program-template()` options from your configuration.

- `prefix()`

  You have to add a trailing dot to the `prefix()`. For example, if `prefix(".sql")` was provided in the configuration, the same option should look like `prefix(".sql.")` in syslog-ng PE 7 LTS.

  If the prefix does not end with a trailing dot, syslog-ng PE 7 LTS will add it automatically, and emit a warning about it.

  The default value has been changed to `.sql.` to reflect this change.

- `read-old-records()`
  - `read-old-records(yes)` works the same way as in syslog-ng PE 6 LTS.
  - `read-old-records(no)`: In syslog-ng PE 6 LTS, whenever syslog-ng PE starts, the application will only read those entries that are created after start. This means between restarts or reload, message loss is possible. In 7 LTS, syslog-ng PE will store the id of the message that was fetched last time in the persist file, even in case of `read-old-records(no)`. When syslog-ng PE starts, it will continue from that `id`. This means that `read-old-records()` only takes effect when syslog-ng PE first starts. After restarts or reloads, `read-old-records()` is ignored.

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 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

- *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.

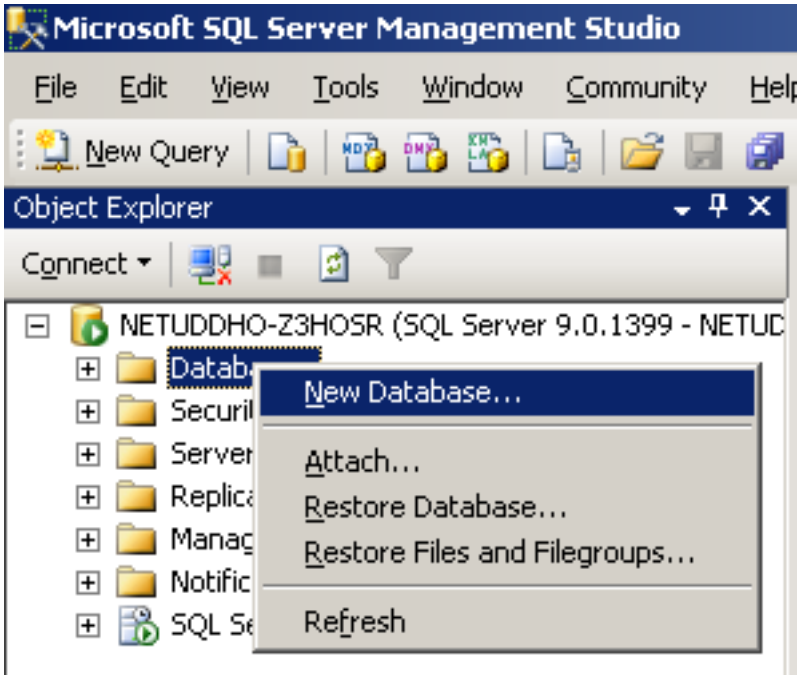
Configuring Microsoft SQL Server to accept logs from syslog-ng

The following describes how to configure your Microsoft SQL Server to enable remote logins and accept log messages from syslog-ng.

To configure your Microsoft SQL Server to enable remote logins and accept log messages from syslog-ng

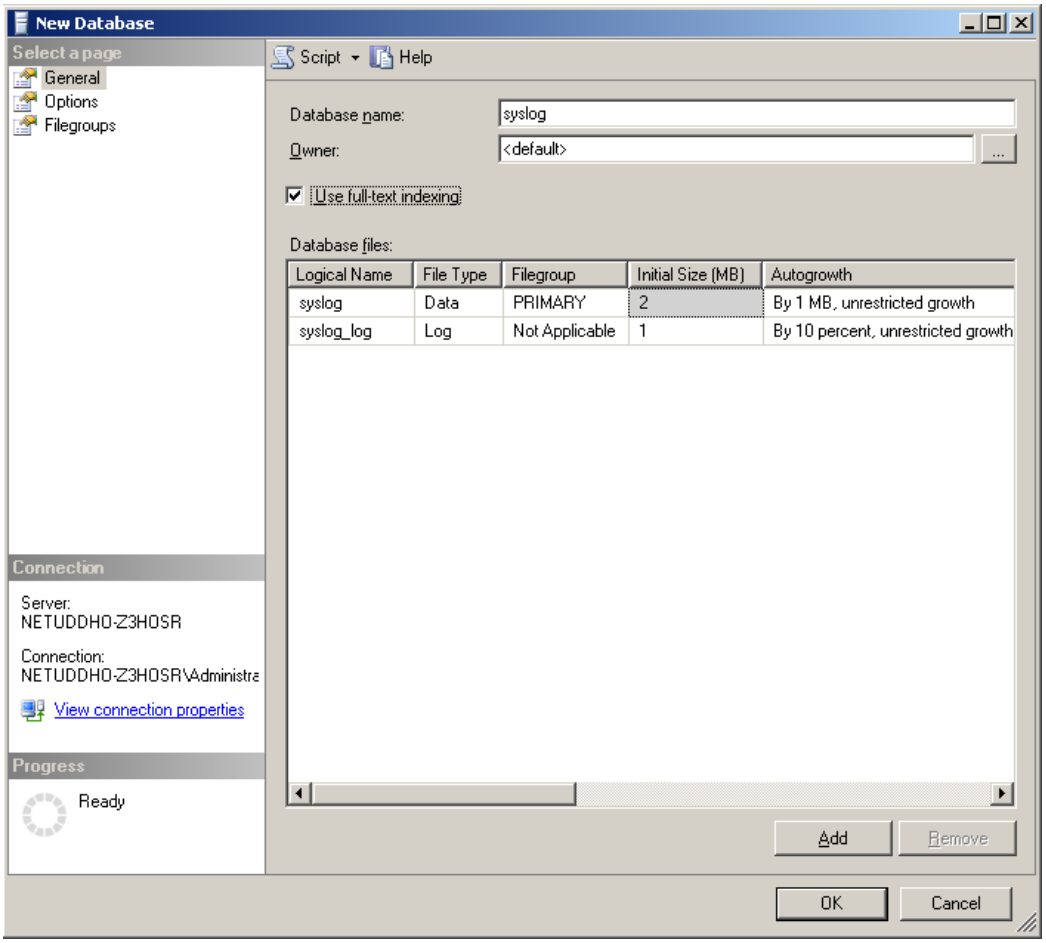
1. Start the SQL Server Management Studio application. Select **Start > Programs > Microsoft SQL Server 2005 > SQL Server Management Studio**.
2. Create a new database.

    a. **Figure 20: Creating a new MSSQL database 1.**



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

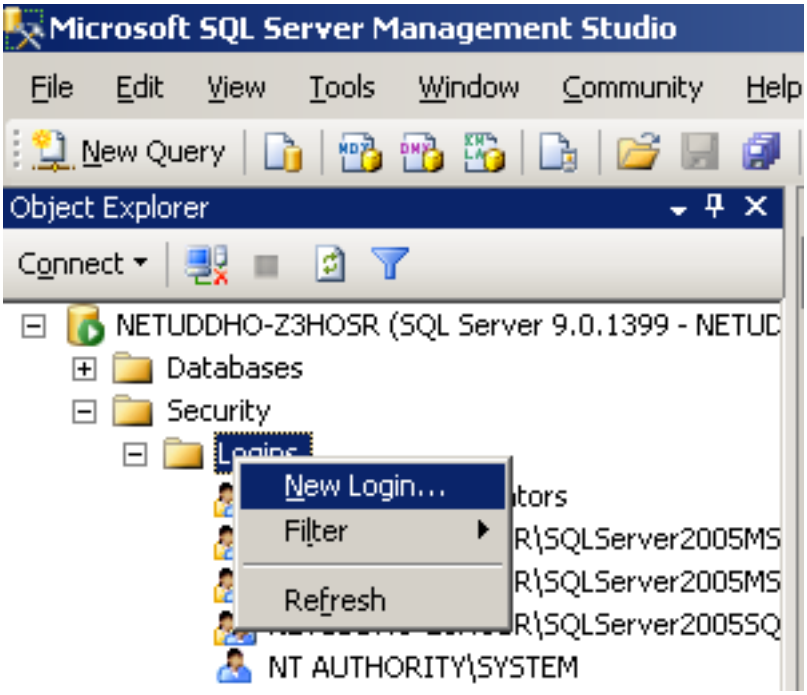
b. **Figure 21: Creating a new MSSQL database 2.**



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

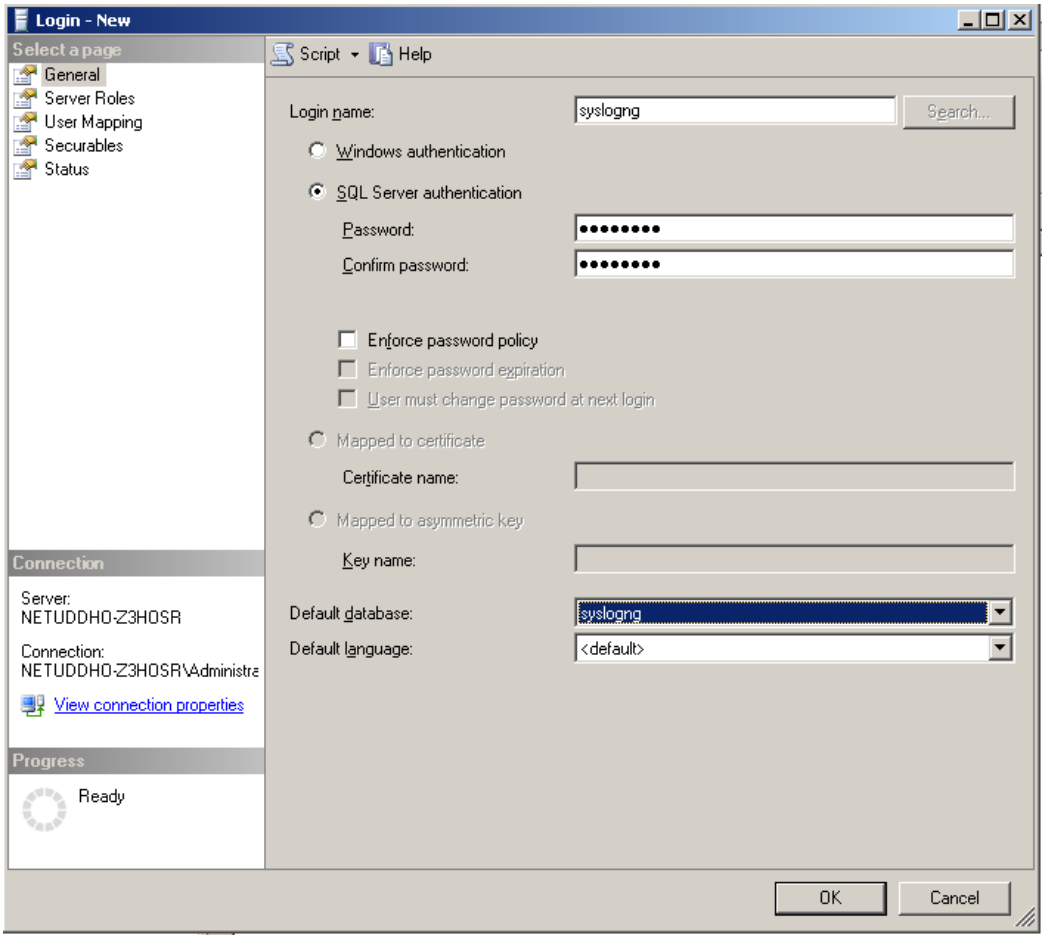
3. Create a new database user and associate it with the new database.

a. **Figure 22: Creating a new MSSQL user 1.**



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

b. **Figure 23: Creating a new MSSQL user 2.**



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

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

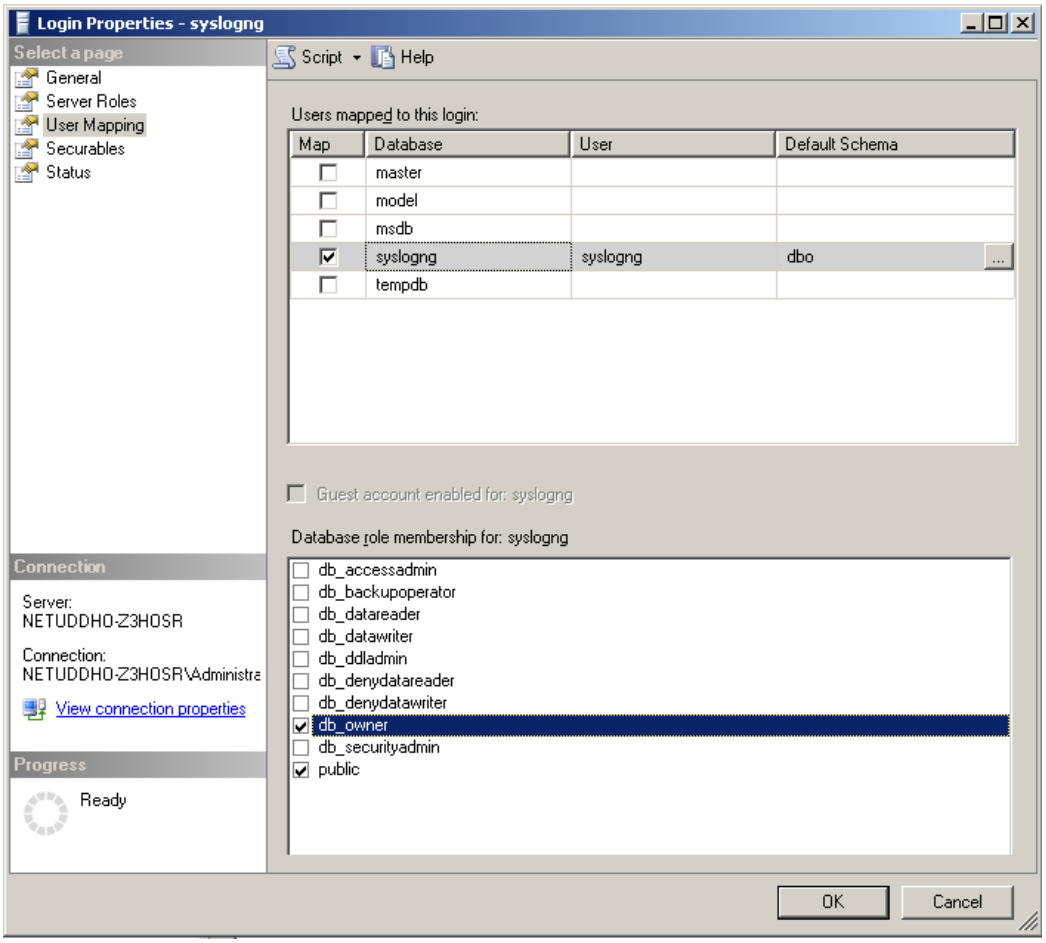
d. In the **Default database** field, select the database created in Step 2 (for example, `syslogng`).

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

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

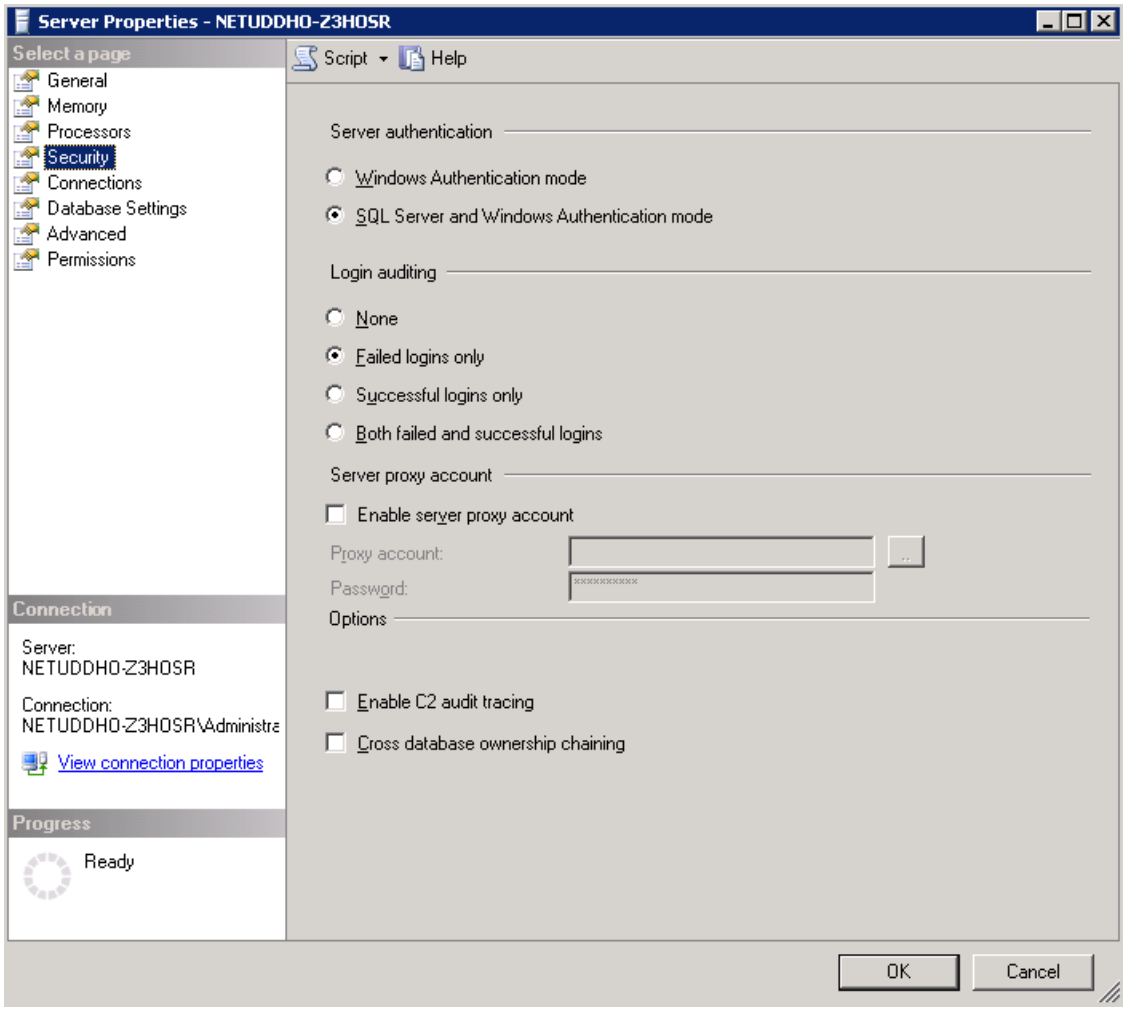
f. In the **Object Explorer**, select **Security > Logins**, then right-click on the new login created in the previous step, and select **Properties**.

g. **Figure 24: Associating database with the new user**



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

4. **Figure 25: Associating database with the new user**



Enable remote logins for SQL users.

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

The syslog-ng PE quick-start guide

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 logserver, see Configuring syslog-ng on client hosts.
- To configure syslog-ng PE as a server that receives log messages from client hosts, see 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 logserver, see Configuring syslog-ng on server hosts.
- For information about managing and checking syslog-ng Premium Edition (syslog-ng PE) services on Linux, see Managing and checking syslog-ng PE service on Linux.

Configuring syslog-ng on client hosts

The following describes how to configure syslog-ng on a client host.

To configure syslog-ng on a client host

1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see Installing syslog-ng PE.
2. 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 system: 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 file: 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")); };
   ```

   Make sure to use a destination that matches the source you configure on your syslog-ng server or relay. For details, see Things to consider when forwarding messages between syslog-ng PE hosts.

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: The default configuration file of syslog-ng PE

The following is the default configuration file of syslog-ng PE7. It collects local log messages and the log messages of syslog-ng PE and saves them in the `/var/log/messages` file.

```
@version: 7.0
@include "scl.conf"
source s_local { system(); internal(); };
destination d_local {
            file("/var/log/messages"); };
log { source(s_local); destination(d_local); };
```

Example: A simple configuration for clients

The following is a simple configuration file that collects local log messages and forwards them to a logserver using the IETF-syslog protocol.

```
@version: 7.0
@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 Troubleshooting syslog-ng for tips on solving common problems.

Configuring syslog-ng on server hosts

The following describes how to configure syslog-ng on a server host.

To configure syslog-ng on a server host

1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see Installing syslog-ng PE.
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 system: 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 syslog: Collecting messages using the IETF syslog protocol (syslog() driver) and tcp, tcp6, udp, udp6: 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 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: 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: 7.0
@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 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.

Configuring syslog-ng on relay hosts

The following describes how to configure syslog-ng on a relay host.

To configure syslog-ng on a relay host

1. Install the syslog-ng application on the host. For details on installing syslog-ng on specific operating systems, see Installing syslog-ng PE.
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. Make sure that you use a destination that matches the source you configured in the previous step. For details, see Things to consider when forwarding messages between syslog-ng PE hosts.
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 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: 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: 7.0
@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 logserver. 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 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`.

  NOTE: The `spoof-source()` option only works under the following conditions:

- The syslog-ng PE binary has been successfully compiled with `--enable-spoof-source`. To check whether the `--enable-spoof-source` option is available on your `syslog-ng-relay`, enter the `syslog-ng --version` command.
- The `spoof-source()` option sends log messages using the highly unreliable UDP transport protocol. One Identity strongly recommends that you consider the risks before using the `spoof-source()` option.

Managing and checking syslog-ng PE service on Linux

This section describes how to start, stop and check the status of syslog-ng Premium Edition (syslog-ng PE) service on Linux.

Starting syslog-ng PE

To start syslog-ng PE, execute the following command as root.

Example: starting syslog-ng PE

```
systemctl start syslog-ng
```

If the service starts successfully, no output will be displayed.

The following message indicates that syslog-ng PE can not start (see Checking syslog-ng PE status):

```
Job for syslog-ng.service failed because the control process exited with error code. See systemctl status syslog-ng.service and journalctl -xe for details.
```

Stopping syslog-ng PE

To stop syslog-ng PE

1. Execute the following command as root.

   Example: command for stopping syslog-ng PE

   ```
   systemctl stop syslog-ng
   ```

2. Check the status of syslog-ng PE service (see Checking syslog-ng PE status).

Restarting syslog-ng PE

To restart syslog-ng PE, execute the following command as root.

Example: command for restarting syslog-ng PE

```
systemctl restart syslog-ng
```

Reloading configuration file without restarting syslog-ng PE

To reload the configuration file without restarting syslog-ng PE, execute the following command as root.

Example: command for reloading the configuration file without restarting syslog-ng PE

```
systemctl reload syslog-ng
```

Checking syslog-ng PE status

To check the following status-related components, observe the suggestions below.

- **Checking the status of syslog-ng PE service**

  To check the status of syslog-ng PE service

  1. Execute the following command as root.

     Example: command for checking the status of syslog-ng PE service

     ```
     systemctl --no-pager status syslog-ng
     ```

  2. Check the `Active:` field, which shows the status of syslog-ng PE service. The following statuses are possible:

- **active (running)** - syslog-ng PE service is up and running

  Example: syslog-ng PE service active

  ```
  syslog-ng.service - System Logger Daemon
  Loaded: loaded (/lib/systemd/system/syslog-ng.service;
  enabled; vendor preset: enabled)
  Active: active (running) since Tue 2019-06-25 08:58:09
  CEST; 5s ago
  Main PID: 6575 (syslog-ng)
  Tasks: 3
  Memory: 13.3M
  CPU: 268ms
  CGroup: /system.slice/syslog-ng.service
  6575 /opt/syslog-ng/libexec/syslog-ng -F --no-caps --
  enable-core
  ```

- **inactive (dead)** - syslog-ng service is stopped

  Example: syslog-ng PE status inactive

  ```
  syslog-ng.service - System Logger Daemon
  Loaded: loaded (/lib/systemd/system/syslog-ng.service;
  enabled; vendor preset: enabled)
  Active: inactive (dead) since Tue 2019-06-25 09:14:16
  CEST; 2min 18s ago
  Process: 6575 ExecStart=/opt/syslog-ng/sbin/syslog-ng -F -
  -no-caps --enable-core $SYSLOGNG_OPTIONS (code=exited,
  status=0/SUCCESS)
  Main PID: 6575 (code=exited, status=0/SUCCESS)
  Status: "Shutting down... Tue Jun 25 09:14:16 2019"
  Jun 25 09:14:31 as-syslog-srv systemd: Stopped System
  Logger Daemon.
  ```

- **Checking the process of syslog-ng PE**

To check the process of syslog-ng PE, execute one of the following commands.

- **Example: command** `ps u `pidof syslog-ng``

  ```
  ps u `pidof syslog-ng`
  ```

  Expected output example:

  USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

  syslogng 6709 0.0 0.6 308680 13432 ? Ss 09:17 0:00 /opt/syslog-ng/libexec/syslog-ng -F --no-caps --enable-core

- **Example: command** `ps axu | grep syslog-ng | grep -v grep`

  ```
  ps axu | grep syslog-ng | grep -v grep
  ```

  Expected output example:

  ```
  syslogng 6709 0.0 0.6 308680 13432 ? Ss 09:17 0:00 /opt/syslog-ng/libexec/syslog-ng -F --no-caps --enable-core
  ```

- **Checking the internal logs of syslog-ng PE**

  The internal logs of syslog-ng PE contains informal, warning and error messages.

  By default, syslog-ng PE log messages (generated on the `internal()` source) are written to **`/var/log/messages`**.

  Check the internal logs of syslog-ng PE for any issue.

- **Message processing**

  The syslog-ng PE application collects statistics about the number of processed messages on the different sources and destinations.

  NOTE: When using `syslog-ng-ctl stats`, consider that while the output is generally consistent, there is no explicit ordering behind the command. Consequently, One Identity does not recommend creating parsers that depend on a fix output order.

  If needed, you can sort the output with an external application, for example, `| sort`.

- **Central statistics**

  To check the central statistics, execute the following command to see the number of received and queued (sent) messages by syslog-ng PE.

  Example: command for checking central statistics

  ```
  watch "/opt/syslog-ng/sbin/syslog-ng-ctl stats | grep ^center"
  ```

  The output will be updated in every 2 seconds.

  If the numbers are changing, syslog-ng PE is processing the messages.

  Example: output example

  ```
  Every 2.0s: /opt/syslog-ng/sbin/syslog-ng-ctl stats | grep
  ^center       Tue Jun 25 10:33:25 2019
  center;;queued;a;processed;112
  center;;received;a;processed;28
  ```

- **Source statistics**

  To check the source statistics, execute the following command to see the number of received messages on the configured sources.

  Example: command for checking central statistics

  ```
  watch "/opt/syslog-ng/sbin/syslog-ng-ctl stats | grep ^source"
  ```

  The output will be updated in every 2 seconds.

  If the numbers are changing, syslog-ng PE is receiving messages on the sources.

Example: output example

```
Every 2.0s: /opt/syslog-ng/sbin/syslog-ng-ctl stats | grep
^source      Tue Jun 25 10:40:50 2019
source;s_null;;a;processed;0
source;s_net;;a;processed;0
source;s_local;;a;processed;90
```

- **Destination statistics**

  To check the source statistics, execute the following command to see the number of received messages on the configured sources.

  Example: command for checking destination statistics

  ```
  watch "/opt/syslog-ng/sbin/syslog-ng-ctl stats | grep ^source"
  ```

  The output will be updated in every 2 seconds.

  If the numbers are changing, syslog-ng PE is receiving messages on the sources.

  Example: output example

  ```
  Every 2.0s: /opt/syslog-ng/sbin/syslog-ng-ctl stats | grep
  ^destination      Tue Jun 25 10:41:02 2019
  destination;d_logserver2;;a;processed;90
  destination;d_messages;;a;processed;180
  destination;d_logserver;;a;processed;90
  destination;d_null;;a;processed;0
  ```

NOTE: If you find error messages in the internal logs, messages are not processed by syslog-ng PE or you encounter any issue, you have the following options:

- Search for the error or issue in our knowledge base.
- Check the following knowledge base articles for further troubleshooting.
- Open a support service request including the results.

The syslog-ng PE configuration file

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. The location of the configuration file depends on how you installed syslog-ng PE. Native packages of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` directory. The `syslog-ng.conf` and `license.txt` files are located under the `/opt/syslog-ng/etc/` directory.

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 7, this line looks like:

```
@version: 7.0
```

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: 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:  7.0

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:

> Alan McKinnon
>
> *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.*

- The main body of the configuration file consists of object definitions: sources, destinations, logpaths 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. Object definitions (also called statements) have the following syntax:

```
type-of-the-object identifier-of-the-object {<parameters>};
```

- *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.

  All identifiers, attributes, and any other strings used in the syslog-ng configuration file are case sensitive.

  > 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.

  > NOTE: Repeating a definition of an object (that is, defining the same object with the same id more than once) is not allowed, unless you use the `@define allow-config-dups 1` definition in the configuration file.
- *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: 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 Global options of syslog-ng PE.

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

- Objects can be used before definition.
- Objects can be defined inline as well. This is useful if you use the object only once (for example, a filter). For details, see Defining configuration objects inline.

- 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.

  Starting with syslog-ng PE version 3.57.0, you can use suffixes for kilo-, mega-, and gigabytes. Use the Kb, Mb, or Gb suffixes for the base-10 version, and Kib, Mib, or Gib for the base-2 version. That is, 2MB means 2000000, while 2MiB means 2097152. For example, to set the `log-msg-size()` option to 2000000 bytes, use `log-msg-size(2Mb)`.
- 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 Regular expressions.
- You can use `if {}`, `elif {}`, and `else {}` blocks to configure conditional expressions. For details, see if-else-elif: Conditional expressions.

Defining configuration objects inline

Starting with syslog-ng PE 3.47.0, you can define configuration objects inline, where they are actually used, without having to define them in a separate object. This is useful if you need an object only once, for example, a filter or a rewrite rule, because it makes the configuration much easier to read. Every object can be defined inline: sources, destinations, filters, parsers, rewrite rules, and so on.

To define an object inline, use braces instead of parentheses. That is, instead of `<object-type> (<object-id>);`, you use `<object-type> {<object-definition>};`

Example: Using inline definitions

The following two configuration examples are equivalent. The first one uses traditional statements, while the second uses inline definitions.

```
source s_local {
    system();
    internal();
};
destination d_local {
    file("/var/log/messages");
```

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

```
log  {
       source  {
             system();
             internal();
       };
       destination  {
             file("/var/log/messages");
       };
 };
```

Using channels in configuration objects

Starting with syslog-ng PE3.47.0, every configuration object is a log expression. Every configuration object is essentially a configuration block, and can include multiple objects. To reference the block, only the top-level object must be referenced. That way you can use embedded log statements, junctions and in-line object definitions within source, destination, filter, rewrite and parser definitions. For example, a source can include a rewrite rule to modify the messages received by the source, and that combination can be used as a simple source in a log statement. This feature allows you to preprocess the log messages very close to the source itself.

To embed multiple objects into a configuration object, use the following syntax. Note that you must enclose the configuration block between braces instead of parenthesis.

```
<type-of-top-level-object>  <name-of-top-level-object>  {
       channel  {
             <configuration-objects>
       };
 };
```

Example: Using channels

For example, to process a log file in a specific way, you can define the required processing rules (parsers and rewrite expressions) and combine them in a single

object:

```
source s_apache {
    channel {
        source { file("/var/log/apache/error.log"); };
        parser(p_apache_parser);
    };
};

log { source(s_apache); ... };
```

The `s_apache` source uses a file source (the error log of an Apache webserver) and references a specific parser to process the messages of the error log. The log statement references only the `s_apache` source, and any other object in the log statement can already use the results of the `p_apache_parser`parser.

NOTE: You must start the object definition with a `channel` even if you will use a `junction`, for example:

```
parser  demo-parser()  {
       channel  {
             junction  {
                   channel  {  ...  };
                   channel  {  ...  };
             };
       };
};
```

If you want to embed configuration objects into sources or destinations, always use channels, otherwise the source or destination will not behave as expected. For example, the following configuration is good:

```
source s_filtered_hosts {
    channel{
        source {
            pipe("/dev/pipe");
            syslog(
                ip(192.168.0.1)
                transport("tcp")
            );
            syslog(
                ip(127.0.0.1)
                transport("tcp")
            );
        };
```

```
        filter {
            netmask(10.0.0.0/16);
        };
    };
};;
```

Global and environmental variables

Starting with syslog-ng PE version 3.24 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: 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 Premium Edition (syslog-ng PE)

To increase its flexibility and simplify the development of additional modules, the syslog-ng PE application is modular. The majority of syslog-ng PE's functionality is in separate modules. As a result, it is also possible to fine-tune 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 that add some functionality to syslog-ng PE (for example, a destination or a source driver).

- To display the list of available modules, run the `syslog-ng --version` command.
- To display the description of the available modules, run the `syslog-ng --module-registry` command.
- To customize which modules syslog-ng PE automatically loads when syslog-ng PE starts, use the `--default-modules` command-line option of syslog-ng PE.
- To request loading a module from the syslog-ng PE configuration file, see 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 The syslog-ng manual page.

Loading modules

The syslog-ng Premium Edition application loads every available module during startup.

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.

NOTE: To disable loading every module automatically, set the `autoload-compiled-modules` global variable to `0` in your configuration file:

```
@define autoload-compiled-modules 0
```

Note that in this case you have to explicitly load the modules you want to use.

Managing complex syslog-ng configurations

The following sections describe some methods that can be useful to simplify the management of large-scale syslog-ng installations.

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 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:

- NOTE: One Identity strongly recommends that you include the `scl.conf` file in your syslog-ng PE configuration. To include the `scl.conf` file in your configuration, use the following syntax:

```
@include "scl.conf"
```

- Defining an object twice is not allowed, unless you use the `@define allow-config-dups 1` definition in the configuration file. 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: 7.0
  @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`. Version 3.77.0 and newer automatically includes the `*.conf` files from the `<directory-where-syslog-ng-is-installed>/scl/*/` directories.
- 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: 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: 7.0
@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: 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 Including configuration files.

Mandatory parameters

You can express in block definitons that a parameter is mandatory by defining it with empty brackets `()`. In this case, the parameter must be overridden in the reference block. Failing to do so will result in an error message and initialization failure.

To make a parameter expand into nothing (for example, because it has no default value, like `hook-commands()` or `tls()`), insert a pair of double quote marks inside the empty brackets: `("")`

Example: Mandatory parameters

The following example defines a TCP source that can receive the following parameters: the port where syslog-ng PE listens (`localport`), and optionally source flags (`flags`).

```
block source my_tcp_source(localport() flags("")) {
      network(port(`localport`) transport(tcp) flags(`flags`));
 };
```

Because `localport` is defined with empty brackets `()`, it is a mandatory parameter. However, the `flags` parameter is not mandatory, because it is defined with an empty double quote bracket pair `("")`. If you do not enter a specific value when referencing this parameter, the value will be an empty string. This means that in this case

```
my_tcp_source(localport(8080))
```

will be expanded to:

```
network(port(8080) transport(tcp) flags());
```

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, use an empty double quote bracket pair `("")` after the name of the argument. To refer the value of the argument in the block, use the name of the argument between backticks (for example, `` `argument1` ``).

Example: 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.

The three dots (…) at the end of the argument list refer to any additional parameters. It tells syslog-ng PE that this macro accepts `__VARARGS__`, therefore any name-value pair can be passed without validation. 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)
      );
};
```

Example: Using arguments in blocks

The following example is the code of the `pacct()` source driver, which is actually a block that can optionally receive two arguments.

```
block source pacct(file("/var/log/account/pacct") follow-freq(1) ...) {
      @module pacctformat
      file("`file`" follow-freq(`follow-freq`) format("pacct") tags
(".pacct") `__VARARGS__`);
 };
```

Generating configuration blocks from a script

The syslog-ng PE application can automatically execute scripts when it is started, and can include the output of such script in the configuration file. The following describes how to create and use a script that generates a part of the syslog-ng PE configuration file (actually, a configuration block). The steps include examples for collecting Apache access log files (`access.log`) from subdirectories, but you can create any script that creates a valid syslog-ng PE configuration snippet.

To create and use a script that generates a part of the syslog-ng PE configuration file (actually, a configuration block)

1. Navigate to the directory where you have installed syslog-ng PE (for example, `/opt/syslog-ng/share/include/scl/`), and create a new directory, for example, `apache-access-logs`. The name of the directory will be used in the syslog-ng PE configuration file as well, so use a descriptive name.
2. Create a file called `plugin.conf` in this new directory.
3. Edit the `plugin.conf` file and add the following line:

   ```
   @module confgen context(source) name(<directory-name>) exec("`scl-root`/<directory-name>/<my-script>")
   ```

   Replace `<directory-name>` with the name of the directory (for example, `apache-access-logs`), and `<my-script>` with the filename of your script (for example, `apache-access-logs.sh`). You can reference the script in your syslog-ng PE configuration file as a configuration block using the value `name` option.

   The `context` option determines the type of the configuration snippet that the script generates, and 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. In the example, `context (source)` means that the output of the script will be used within a source statement.
4. Write a script that generates the output you need, and formats it to a configuration snippet that syslog-ng PE can use. The filename of the script must match with the filename used in `plugin.conf`, for example, `apache-access-logs.sh`.

The following example checks the `/var/log/apache2/` directory and its subdirectories, and creates a source driver for every directory that contains an `access.log` file.

```
#!/bin/bash
 for i in `find /var/log/apache2/ -type d`; do
       echo "file(\"$i/access.log\" flags(no-parse) program_override
(\"apache2\"));";
 done;
```

The script generates an output similar to this one, where `service*` is the actual name of a subdirectory:

```
file("/var/log/apache2/service1/access.log"
       flags(no-parse)
       program_override("apache2")
 );
 file("/var/log/apache2/service2/access.log"
       flags(no-parse)
       program_override("apache2")
 );
```

5. Include the `plugin.conf` file in the `syslog-ng.conf` file — or a file already included into `syslog-ng.conf`. Version 3.77.0 and newer automatically includes the `*.conf` files from the `<directory-where-syslog-ng-is-installed>/scl/*/` directories. For details on including configuration files, see Including configuration files.
6. Add the block you defined in the `plugin.conf` file to your syslog-ng PE configuration file. You can reference the block using the value of the `name` option from the `plugin.conf` file, followed by parentheses, for example, `apache-access-logs()`. Make sure to use the block in the appropriate context of the configuration file, for example, within a source statement if the value of the `context` option in the `plugin.conf` file is source.

```
@include "scl.conf"
...
source s_apache {
    file("/var/log/apache2/access.log"
        flags(no-parse)
        program_override("apache2")
    );
    file("/var/log/apache2/error.log"
        flags(no-parse)
        program_override("apache2")
    );
    file("/var/log/apache2/ssl.log"
        flags(no-parse)
        program_override("apache2")
    );
```

```
    apache-access-logs();
};

log {
    source(s_apache);
    destination(d_central);
};
...
```

7. Check if your modified syslog-ng PE configuration file is syntactically correct using the `syslog-ng --syntax-only` command.
8. If your modified configuration is syntactically correct, load the new configuration file using the `syslog-ng-ctl reload` command.

Python code in external files

You can extend and customize syslog-ng PE easily by writing destinations, parsers, template functions, and sources in Python.

Instead of writing Python code into your syslog-ng PE configuration file, you can store the Python code for your Python object in an external file. That way, it is easier to write, maintain, and debug the code. You can store the Python code in any directory in your system, but make sure to include it in your Python path.

When referencing a Python class from an external file in the `class()` option of a Python block in the syslog-ng PE configuration file, the class name must include the name of the Python file containing the class, without the path and the .py extension. For example, if the MyDestination class is available in the `/etc/syslog-ng/etc/pythonexample.py` file, use `class ("pythonexample.MyDestination")`:

```
destination  d_python_to_file  {
       python(
             class("pythonexample.MyDestination")
       );
};
log  {
       source(src);
       destination(d_python_to_file);
};
```

If you store the Python code in a separate Python file and only include it in the syslog-ng PE configuration file, make sure that the PYTHON_PATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start syslog-ng PE manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`

In production, when syslog-ng PE starts on boot, you must configure your startup script to include the Python path. The exact method depends on your operating system. For recent Red Hat Enterprise Linux, Fedora, and CentOS distributions that use systemd, the `systemctl` command sources the `/etc/sysconfig/syslog-ng` file before starting syslog-ng PE. (On openSUSE and SLES, `/etc/sysconfig/syslog` file.) Append the following line to the end of this file: `PYTHONPATH="<path-to-your-python-file>"`, for example, `PYTHONPATH="/opt/syslog-ng/etc"`

To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of syslog-ng PE. For details, see Logging from your Python code.

Logging from your Python code

You can extend and customize syslog-ng PE easily by writing destinations, parsers, template functions, and sources in Python.

To debug and troubleshoot your Python code, syslog-ng PE allows you to use the `logger()` method to send log messages to the `internal()` source of syslog-ng PE. That way the diagnostic messages of your Python code are treated the same way as other such log messages of syslog-ng PE. This has the following benefits:

- The `logger()` method respects the log level settings of syslog-ng PE. You can write error, warning, info, debug, and trace level messages.
- You can follow what your Python code is doing even if syslog-ng PE is running as a daemon in the background.

Logging to the `internal()` source is available in syslog-ng PE version 3.207.0.14 and later.

To send log messages to the internal() source from Python

1. Add the following import to your Python code:

   ```
   import syslogng
   ```

2. Create a logger object:

   ```
   logger = syslogng.Logger()
   ```

3. Use the logger object in your Python code, for example:

   ```
   logger.info("This is a sample log message send from the Python code.")
   ```

   You can use the following log levels: `logger.error`, `logger.warning`, `logger.info`, `logger.debug`, `logger.trace`

4. Make sure that your syslog-ng PE configuration includes the `internal()` source, for example:

```
source s_internal { internal(); };
destination d_internal { file("/var/log/internal.txt"); };
log {source(s_internal); destination(d_internal); };
```

Collecting log messages — sources and source drivers

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: 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: 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: 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 Routing messages: log paths, flags, 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 7: 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 multithreaded IPC method called door. By default the door used by `syslogd` is `/etc/.syslog_door`.
HP-UX 11 or later	HP-UX uses a named pipe called `/dev/log` that is padded to 2048 bytes, for example, `source s_hp-ux {pipe ("/dev/log" pad-size(2048)}`.

Platform	Method
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: 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 8: Source drivers available in syslog-ng

Name	Description
file()	Opens the specified file and reads messages.
internal()	Messages generated internally in syslog-ng.
linux-audit()	Reads the logfiles of the auditd application.
network()	Receives messages from remote hosts using the BSD-syslog protocol over IPv4 and IPv6. Supports the TCP, UDP, ALTP, 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.
python() and python-fetcher	Receive or fetch messages using a custom source written in Python.

Name	Description
()	
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.
windowsevent()	Reads messages from the Windows Event Collector tool.

`default-network-drivers`: Receive and parse common syslog messages

The `default-network-drivers()` source is a special source that uses multiple source drivers to receive and parse several different types of syslog messages from the network. Available in version 7.0.93.16 and later.

To use the `default-network-drivers()` source, the `scl.conf` file must be included in your syslog-ng PE configuration:

```
@include "scl.conf"
```

Also, make sure that your SELinux, AppArmor, and firewall settings permit syslog-ng Premium Edition to access the ports where you want to receive messages, and that no other application is using these ports. By default, the `default-network-drivers()` source accepts messages on the following ports:

- 514, both TCP and UDP, for RFC3164 (BSD-syslog) formatted traffic
- 601 TCP, for RFC5424 (IETF-syslog) formatted traffic
- 6514 TCP, for TLS-encrypted traffic

In addition to receiving messages on different ports and in different formats, this source tries to parse the messages automatically. If successful, it sets the `${.app.name}` name-value pair to the name of the application that sent the log message. Currently it uses the following procedures.

CAUTION:

If you do not configure the TLS keys to dislay to the clients, syslog-ng PE cannot accept encrypted connections. The application starts and listens on TCP:6514, and can receive messages on other ports, but will display a warning messages about missing keys.

Parsing RFC3164-formatted messages

For RFC3164-formatted messages (that is, messages received on the ports set in options `udp-port()` and `tcp-port()` which default to port 514), syslog-ng PE attempts to use the following parsers. If a parser cannot parse the message, it passes the original message to the next parser.

1. Parse the incoming raw message as a message from a Cisco device.
2. Parse the incoming message as an RFC3164-formatted message.
    - If the incoming message was sent by a syslog-ng PE client using the syslog-ng () destination, parse its fields as a syslog-ng message.

      The Enterprise-wide message model or EWMM allows you to deliver structured messages from the initial receiving syslog-ng component right up to the central log server, through any number of hops. It does not matter if you parse the messages on the client, on a relay, or on the central server, their structured results will be available where you store the messages. Optionally, you can also forward the original raw message as the first syslog-ng component in your infrastructure has received it, which is important if you want to forward a message for example, to a SIEM system. To make use of the enterprise-wide message model, you have to use the syslog-ng() destination on the sender side, and the default-network-drivers() source on the receiver side.
    - Otherwise, apply the application adapters if the message was sent from an application that already has a specific parser in syslog-ng PE (for example, Splunk Common Information Model (CIM), iptables, or sudo).

Parsing RFC5424-formatted messages

For RFC5424-formatted messages (that is, messages received on the ports set in options `rfc5424-tls-port()` and `rfc5424-tcp-port()`, which default to port 6514 and 601), syslog-ng PE parses the message according to RFC5424, then attempts apply the application adapters if the message was sent from an application that already has a specific parser in syslog-ng PE (for example, Splunk Common Information Model (CIM), iptables, or sudo).

Example: Using the default-network-drivers() driver

The following example uses only the default settings.

```
source  s_network  {
       default-network-drivers();
 };
```

The following example can receive TLS-encrypted connections on the default port (port 6514).

```
source  s_network  {
       default-network-drivers(
             tls(
                   key-file("/path/to/ssl-private-key")
                   cert-file("/path/to/ssl-cert")
                )
       );
 };
```

default-network-drivers() source options

The `systemd-journal()` driver has the following options.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.

- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

log-msg-size()

Type:	number (bytes)
Default:	Use the global `log-msg-size()` option, which defaults to `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections.

Note that the total number of connections the `default-network-drivers()` source can use is 3*`max-connections()`, because this value applies to the `network(tcp)`, `syslog(tcp)`, and `syslog(tls)` connections individually.

rfc5424-tcp-port()

Type:	number
Default:	601

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Description: The TCP port number where the `default-network-drivers()` source receives RFC5424-formatted (IETF-syslog) messages.

rfc5424-tls-port()

Type:	number
Default:	6514

Description: The TCP port number where the `default-network-drivers()` source receives RFC5424-formatted (IETF-syslog), TLS-encrypted messages.

CAUTION:

To receive messages using a TLS-encrypted connection, you must set the `tls(key-file() cert-file())` options of the `default-network-drivers()` source. For example:

```
source s_network {
    default-network-drivers(
        tls(
            key-file("/path/to/ssl-private-key")
            cert-file("/path/to/ssl-cert")
        )
    );
};
```

tcp-port()

Type:	number
Default:	514

Description: The TCP port number where the `default-network-drivers()` source receives RFC3164-formatted (BSD-syslog) messages.

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 TLS options.

udp-port()

Type:	number
Default:	514

Description: The UDP port number where the `default-network-drivers()` source receives RFC3164-formatted (BSD-syslog) messages.

`internal`: 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: 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
Default:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with flow-control.

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.

`file`: Collecting messages from text files

Collects log messages from plain-text files, for example, from the logfiles of an Apache webserver. If you want to use wildcards in the filename, use the `wildcard-file()` source.

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. If you want to use wildcards in the filename, use the `wildcard-file()` source. For the list of available optional parameters, see file() source options.

CAUTION:

Hazard of data loss! If your log files are on an NFS-mounted network file system, see Using syslog-ng PE with NFS or CIFS (or SMB) file system for log files.

Declaration

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

Example: Using the file() driver

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

Example: 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.

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() 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 Message size and encoding.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.

- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.

- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow-freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically. 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
Default:	10

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

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:	10000

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()`.

log-msg-size()

Type:	number (bytes)
Default:	Use the global `log-msg-size()` option, which defaults to `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program-override()` instead.

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.

To use the `multi-line-garbage()` option, set the `multi-line-mode()` option to `prefix-garbage`.

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()`.

multi-line-mode()

Type:	indented|regexp
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The syslog-ng PE application provides the following methods to process multi-line messages: `multi-line-mode(indented)`, and `multi-line-mode(prefix-garbage)`.

- The *indented* mode can process messages where each line that belongs to the previous line is indented by whitespace, and the message continues until the first non-indented line. For example, the Linux kernel (starting with version 3.5) uses this format for `/dev/log`, as well as several applications, like Apache Tomcat.

Example: Processing indented multi-line messages

```
source  s_tomcat  {
       file("/var/log/tomcat/xxx.log"
             multi-line-mode(indented)
       );
 };
```

- The *prefix-garbage* mode uses a string or regular expression (set in `multi-line-prefix()`) that matches the beginning of the log messages, 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. For details on using `multi-line-mode(prefix-garbage)`, see the `multi-line-prefix()` and `multi-line-garbage()` options.
- The `prefix-suffix` mode uses a string or regular expression (set in `multi-line-prefix()`) that matches the beginning of the log messages, ignores newline characters from the source until a line matches the regular expression set in `multi-line-suffix()`, and treats the lines between `multi-line-prefix()` and `multi-line-suffix()` as a single message. Any other lines between the end of the message and the beginning of a new message (that is, a line that matches the `multi-line-prefix()` expression) are discarded. For details on using `multi-line-mode(prefix-suffix)`, see the `multi-line-prefix()` and `multi-line-suffix()` options.

  The `prefix-suffix` mode is similar to the `prefix-garbage` mode, but it appends the garbage part to the message instead of discarding it.

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 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.

multi-line-prefix()

Type:	regular expression starting with the ^ character
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 (always start with the ^ character). 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.

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 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: 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-mode(regexp) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]
{2}\.") flags(no-parse));};
 };
```

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

multi-line-suffix()

Type:	regular expression
Default:	empty string

Description: Use the `multi-line-suffix()` option when processing multi-line messages. Specify a string or regular expression that matches the end of the multi-line message.

To use the `multi-line-suffix()` option, set the `multi-line-mode()` option to `prefix-suffix`. See also the `multi-line-prefix()` option.

multi-line-timeout()

Type:	number [seconds]
Default:	`[0 == disable]`

Description: If any `multi-line` option is enabled, there are two possible scenarios:

- the requirement specified in any `multi-line` option is met, which results in syslog-ng PE sending the full message (even multi-line messages)
- the number of seconds specified in `multi-line-timeout()` passes, which results in syslog-ng PE sending the current message, which could be partial

NOTE: One Identity recommends that you set `multi-line-timeout()` to a higher value than (preferably a multiple of) `follow-freq()`.

The recommended minimum value is 10 seconds.

pad-size()

Type:	number
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. 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.

google-pubsub: collecting messages from the Google Pub/Sub messaging service

From version 7.0.22, syslog-ng Premium Edition (syslog-ng PE) can collect messages from the Google Pub/Sub messaging service using the `google-pubsub()` source.

NOTE: The rest of this section and its subsections assume that you are familiar with the Google Pub/Sub messaging service, and its concepts and terminology.

For more information about Google Pub/Sub's messaging service, see What Is Pub/Sub?.

For more information about setting up your Google Pub/Sub messaging service system, see Quickstart: building a functioning Pub/Sub system.

For more information about how the syslog-ng PE application sends logs to the Google Pub/Sub messaging service through the `google_pubsub()` destination, see google_pubsub(): Sending logs to the Google Cloud Pub/Sub messaging service.

CAUTION:

This is a Preview Feature, which provides an insight to planned enhancements to functionality in the product. Consider this Preview Feature a work in progress, as it may not represent the final design and functionality.

This feature has completed QA release testing, but its full impact on production systems has not been determined yet, and potential future changes in functionality and the user interface may result in compatibility issues in your current settings.

One Identity recommends the following:

- **Consider the potential risks when using this functionality in a production environment.**
- **Consider the Support Policy on Product Preview Features before using this functionality in a production environment.**
- **Closely and regularly keep track of official One Identity announcements about potential changes in functionality and the user interface. If these potential changes affect your configuration, check the changes you have to make in your configuration, otherwise your syslog-ng PE application may not start after upgrade.**
- **Always perform tests prior to upgrades in order to avoid the risks mentioned.**

However, you are welcome to try this feature and if you have any feedback, Contact One Identity.

Support Policy on Product Preview Features

The One Identity Support Team will:

- **Accept and review each service request opened regarding a Preview Feature.**
- **Consider all service requests relating to a Preview Features as severity level 3.**
- **Provide best effort support to resolve any issues relating to a Preview Feature.**
- **Work with customers to log any product defects or enhancements relating to Preview Features.**
- **Not accept requests for escalations regarding Preview Features.**
- **Not provide after-hours support for Preview Features.**

Prerequisites

To configure the `google-pubsub()` source on syslog-ng Premium Edition (syslog-ng PE), you must have each of the following:

Google Cloud Platform side prerequisites

- A Google Cloud project in your Google Cloud Platform (for example, `syslog-ng-pubsub-src`).

  For more information about creating a Google Cloud project, see Creating and managing projects.
- Appropriate credentials through a Google Cloud Pub/Sub messaging service account.

  For more information, see Create service account credentials.
- A Google Pub/Sub topic and a Google Pub/Sub subscription.

  For more information, see Set up your Google Cloud project and Pub/Sub topic and subscriptions.

Google Pub/Sub Subscription prerequisites

- **Subscription name** (configured previously, see Set up your Google Cloud project and Pub/Sub topic and subscriptions)
- **Topic name** (configured previously, see Set up your Google Cloud project and Pub/Sub topic and subscriptions)
- **Delivery type**
- **Subscription expiration**
- **Acknowledgement deadline**

  NOTE: In case you encounter message duplication, One Identity recommends that you check the following parameters in your configuration:

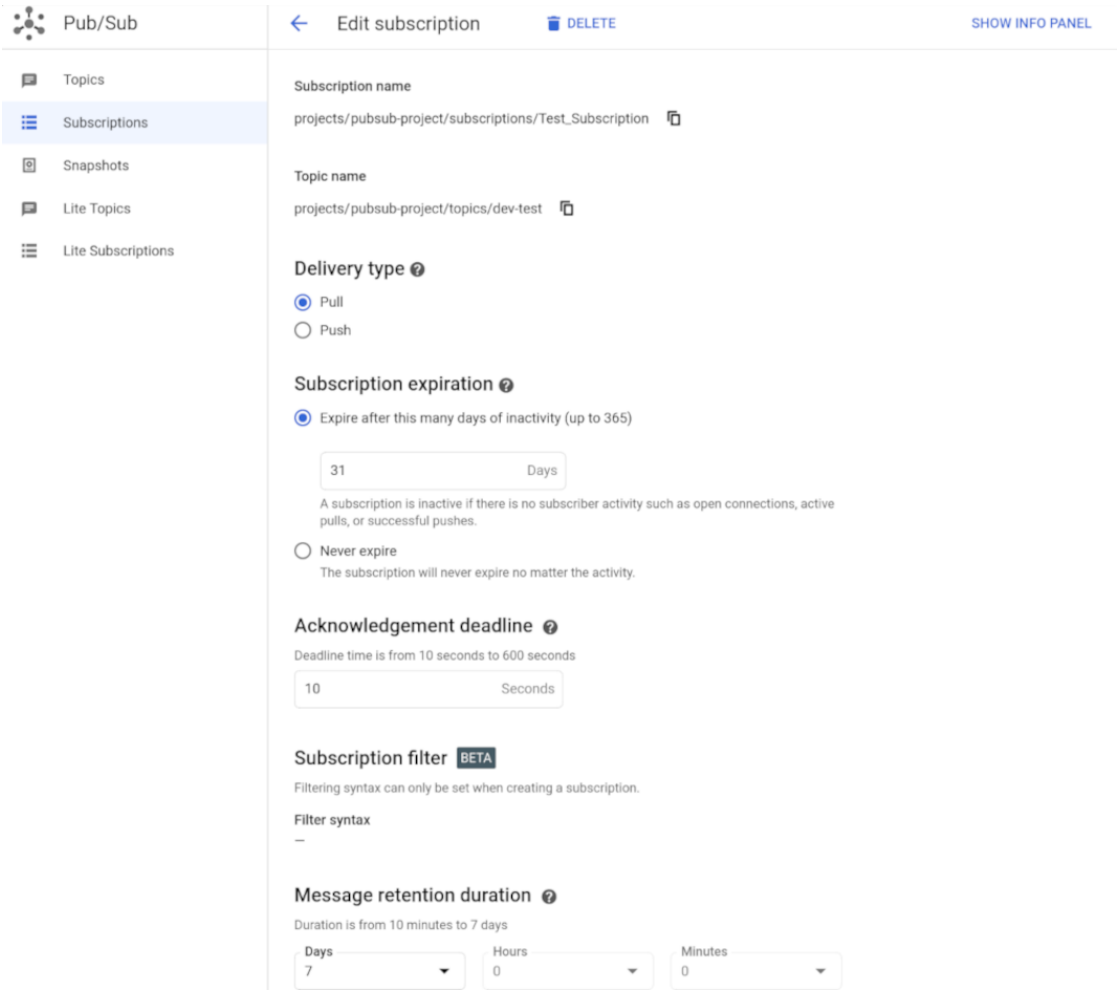
  - `log-fetch-limit()`: if it set to a high value, it is possible that syslog-ng PE can not process the fetched messages within the configured **Acknowledgement deadline**.
  - `ack-tracker-timeout()`: if it set to a higher value than the **Acknowledgement deadline** configured on the Google Cloud Platform, set it to a lower value.

  For more information about message duplication, see Preventing message duplication resulting from the At-Least-Once delivery behavior.
- **Subscription filter**
- **Message retention duration**

Example: Google Pub/Sub

The following example shows a Google Pub/Sub with all necessary parameters configured.

Figure 26: A Google Pub/Sub subscription with all necessary parameters configured



Limitations

The current implementation of the `google-pubsub()` source has the following limitations:

- The At-Least-Once delivery behavior takes effect after creating your subscription.

  NOTE: The At-Least-Once delivery behavior (which means that if an error occurs, it is more acceptable to duplicate messages than to lose any of them) only takes

effect after you create your Google Pub/Sub subscription.

The At-Least-Once delivery behavior is intentional, and its purpose is to avoid potential log loss.

For more information, see Preventing message duplication resulting from the At-Least-Once delivery behavior.

- The Google Pub/Sub service cannot guarantee message ordering.

  As a result of the At-Least-Once delivery behavior, messages may be delivered out of order, especially if an outstanding message is not acknowledged by the subscriber before the **Acknowledgement deadline** passes. In this case, the Google Pub/Sub service will attempt to redeliver the message. As a result, the Google Pub/Sub service cannot guarantee message ordering.

- The syslog-ng PE application retains acknowledgements on the source side and either acknowledges an `ack-tracker-batch-size()` number of messages in a batch, or sends acknowledgements after the `ack-tracker-timeout()` expires. If the value of your `ack-tracker-timeout()` is larger than the value of your **Acknowledgement deadline**, it may result in message duplication.

  NOTE: In case you encounter message duplication, One Identity recommends that you check the following parameters in your configuration:

  - `log-fetch-limit()`: if it set to a high value, it is possible that syslog-ng PE can not process the fetched messages within the configured **Acknowledgement deadline**.
  - `ack-tracker-timeout()`: if it set to a higher value than the **Acknowledgement deadline** configured on the Google Cloud Platform, set it to a lower value.

  For more information about message duplication, see Preventing message duplication resulting from the At-Least-Once delivery behavior.

Supported platforms

The current implementation of the `google-pubsub()` source works on all supported syslog-ng PE 7 LTS platforms.

Declaration

You can use the following example as a configuration block template for declaring the `google-pubsub()` source in your configuration:

```
google-pubsub(project("project")
    subscription("sub")
    credentials("creds.json")
 );
```

The Google Pub/Sub message format in syslog-ng PE

From version 7.0.22, syslog-ng Premium Edition (syslog-ng PE) can collect messages from Google Pub/Sub.

NOTE: The rest of this section assumes that you are familiar with the Google Pub/Sub messaging service, and its concepts and terminology.

Messages on the Google Cloud Platform side

The syslog-ng PE application's `google-pubsub()` source collects Google Pub/Sub messages in a format that has two message parts (**Message body** and **Message attributes**) on the Google Cloud Platform side.

NOTE: Google Pub/Sub messages must contain at least one of the **Message body** and **Message attributes** message parts.

For more information about Google Pub/Sub message format on the Google Cloud Platform side, see Message format.

For more information about publishing Google Pub/Sub messages on the Google Cloud Platform side, see Publishing messages.

Processing the Message body contents on the syslog-ng PE side

After collecting the contents of the Google Pub/Sub message's **Message body** field as raw, unformatted data, the `google-pubsub()` source stores the message contents in the `$MESSAGE` syslog-ng PE macro. Next, syslog-ng PE prepends a header to the message contents, and the resulting data will form the syslog-ng PE output.

Example: incoming Google Pub/Sub Cloud message contents, resulting `$MESSAGE` macro contents, and processed output message contents with a prepended Message header

Incoming message contents on the Google Pub/Sub Cloud side:

```
<38>Feb 25 14:09:07 testhost testapp: test message - 1
```

The contents of the relevant `$MESSAGE` macro:

```
* name='MESSAGE', value='<38>Feb 25 14:09:07 testhost testapp: test
message - 1'
```

By default, the syslog-ng PE application prepends a Message header to the `$MESSAGE` macro contents to form an output with a similar structure:

```
<13>Sep 29 14:59:28 ubuntu-xenail-amd64 <38>Feb 25 14:09:07 testhost
testapp: test message - 1
```

The contents of the Google Pub/Sub Message body on the syslog-ng Premium Edition (syslog-ng PE) side

The syslog-ng PE application's `google-pubsub()` source collects Google Pub/Sub messages in a format that has two message parts (**Message body** and **Message attributes**) on the Google Cloud Platform side.

NOTE: Google Pub/Sub messages must contain at least one of the **Message body** and **Message attributes** message parts.

After collecting the contents of the Google Pub/Sub message's **Message body** field as raw, unformatted data, the `google-pubsub()` source stores the message contents in the `$MESSAGE` syslog-ng PE macro. Next, syslog-ng PE prepends a header to the message contents, and the resulting data will form the syslog-ng PE output.

If the Google Pub/Sub message also contains data from **Message attributes**, these attributes are displayed in the output as attribute names following the prepended prefix.

Example: Message body contents in the $MESSAGE syslog-ng PE macro

The following example shows illustrates what **Message body** contents on the Google Pub/Sub Cloud side will look like in the `$MESSAGE` syslog-ng PE macro:

With the incoming **Message body** contents on the Google Pub/Sub Cloud side:

```
<38>Feb 25 14:09:07 testhost testapp: test message - 1
```

The contents of the relevant `$MESSAGE` macro will look similar to this:

```
* name='MESSAGE', value='<38>Feb 25 14:09:07 testhost testapp: test
message - 1'
```

The contents of the Google Pub/Sub Message attributes on the syslog-ng PE (syslog-ng PE) side

The syslog-ng PE application's `google-pubsub()` source collects Google Pub/Sub messages in a format that has two message parts (**Message body** and **Message attributes**) on the Google Cloud Platform side.

NOTE: Google Pub/Sub messages must contain at least one of the **Message body** and **Message attributes** message parts.

The contents of the message attributes can be customized with the prefix() option, where the prefix is `.pubsub.` by default.

The contents of the message attributes message part are key-value pairs, for example, `${.pubsub.attribute_name}`.

Example: incoming Google Pub/Sub Cloud message contents (including message attributes), the resulting $MESSAGE macro contents and key-values, and processed output message contents with a prepended Message header

Incoming message contents on the Google Pub/Sub Cloud side:

```
<38>Feb 25 14:09:07 testhost testapp: test message - 1
```

Incoming attribute values to the message on the Google Pub/Sub Cloud side:

```
* attrib_name1="attrib_value1"
* attrib_name2="attrib_value2"
```

The contents and key-values of the relevant `$MESSAGE` macros:

```
* name='MESSAGE', value='<38>Feb 25 14:09:07 testhost testapp: test
message - 1'
* attrib_name1="attrib_value1"
* attrib_name2="attrib_value2"
```

By default, the syslog-ng PE application prepends a Message header to the `$MESSAGE` macro contents to form an output with a similar structure:

```
<13>Sep 29 14:59:28 ubuntu-xenail-amd64 <38>Feb 25 14:09:07 testhost
testapp: test message - 1
```

Processing incoming message contents in raw message format and in .JSON format

Depending on the format of your incoming message contents, and whether you want to forward these contents in .JSON format instead of a raw message format, you may have to filter, parse, or otherwise modify your incoming message contents.

The following examples illustrate what incoming message contents look like in raw message format and in .JSON message format, and how to handle and process message contents in .JSON format.

Processing incoming message contents in raw message format

If your incoming message contents are in a raw message format, and you do not filter or otherwise modify the contents, syslog-ng PE will automatically forward the contents in the same raw message format, and the output will look similar to this:

Example: incoming Google Pub/Sub Cloud message contents (in raw message format), the resulting $MESSAGE macro contents, and the processed output message contents with a prepended Message header

Incoming message contents in raw message format on the Google Pub/Sub Cloud side:

```
<38>Feb 25 14:09:07 testhost testapp: test message mytestmessage
```

The contents of the relevant `$MESSAGE` macros:

```
* name='MESSAGE', value='{"data":"<38>Feb 25 14:09:07 testhost testapp:
test message mytestmessage"}'
```

By default, the syslog-ng PE application prepends a Message header to the `$MESSAGE` macro contents to form an output with a similar structure:

```
<13>Sep 29 17:03:58 ubuntu {"data":"<38>Feb 25 14:09:07 testhost testapp:
test message mytestmessage"}\x0a
```

Processing incoming message contents in .JSON message format

The syslog-ng PE application's `google-pubsub()` source collects Google Pub/Sub messages in a format that has two message parts (**Message body** and **Message attributes**) on the Google Cloud Platform side.

However, depending on how you configure the Google Pub/Sub messaging service on the Platform side, you may receive incoming messages in .JSON format.

Configuring syslog-ng PE to process incoming .JSON message formats

Even if your incoming message contents are originally in .JSON format, syslog-ng PE will store them in a raw message format.

If you want to forward your message contents in a .JSON format along with message attributes, you can use `format-json()` as a rewrite rule or as a destination template.

Example: configuring syslog-ng PE to transform raw incoming message format to .JSON message format using format-json()

The following configuration example illustrates how you can use `format-json()` to configure syslog-ng PE to transform the raw message format of the incoming message to a .JSON format message that contains the contents of both the **Message body** and the **Message attributes**.

```
log {
    source {
       google-pubsub(project("syslog-ng-pubsub-src") subscription("sub")
credentials("syslog-ng-pubsub-creds.json"));
    };

    if {
       parser { json-parser( prefix(".gpub.data.") template
("$MESSAGE")); };
    }
    else {
       rewrite { set("$MESSAGE" value(".gpub.data")); };
    };

    destination {
       file("/tmp/output" template("$(format-json --key .pubsub.* --
shift 8   --key .gpub.* --shift   6)\n"));
    };
 };
```

In this case, the incoming message contents on the Google Pub/Sub Platform side are the following:

```
{"message_body_json_field1":  "value1",  "message_body_json_field2":
"value2"}
```

The Pub/Sub attributes of the message are the following:

```
pubsubmsgattribute1
pubsubmsgattribute2
```

The output message looks like this:

```
{"pubsubmsgattribute2":"pubsubattrvalue2","pubsubmsgattribute1":"pubsubatt
rvalue1","data":{"message_body_json_field2":"value2","message_body_json_
field1":"value1"}
```

google-pubsub() source options

The `google-pubsub()` source has the following options.

Required parameters

- `credentials()`
- `project()`
- `subscription()`

Optional parameters

- `ack-tracker-batch-size()`
- `ack-tracker-timeout()`
- `log-fetch-limit()`
- `prefix()`
- `time-reopen()`
- `workers()`

The `google-pubsub()` source options, in more detail:

ack-tracker-batch-size()

Type:	string
Default:	100
Required:	no

Description: Optional parameter.

The syslog-ng PE application retains acknowledgements on the source side and either acknowledges an `ack-tracker-batch-size()` number of messages in a batch, or sends acknowledgements after the `ack-tracker-timeout()` expires. If the value of your `ack-tracker-timeout()` is larger than the value of your **Acknowledgement deadline**, it may result in message duplication.

ack-tracker-timeout()

Type:	time [milliseconds]
Default:	3000
Required:	no

Description: Optional parameter.

The syslog-ng PE application retains acknowledgements on the source side and either acknowledges an `ack-tracker-batch-size()` number of messages in a batch, or sends acknowledgements after the `ack-tracker-timeout()` expires. If the value of your `ack-tracker-timeout()` is larger than the value of your **Acknowledgement deadline**, it may result in message duplication.

credentials()

Type:	string
Default:	n/a
Required:	yes

Description: Required parameter.

The credentials of your Google Pub/Sub project.

log-fetch-limit()

Type:	number
Default:	100
Required:	no

Description: Optional parameter.

The maximum number of messages fetched from a source during a single poll loop.

prefix()

Type:	string
Default:	`.pubsub.`
Required:	no

Description: Optional parameter.

This prefix will be added to the name of the macros created from the message attributes of the Google Pub/Sub message.

project()

Type:	string
Default:	n/a
Required:	yes

Description: Required parameter.

The ID of your Google Pub/Sub project.

subscription()

Type:	string
Default:	n/a
Required:	yes

Description: Required parameter.

The ID of your Google Pub/Sub subscription.

time-reopen()

Type:	number (seconds)
Default:	60
Required:	no

Description: Optional parameter.

The time to wait in seconds before a broken connection is reestablished.

workers()

Type:	integer
Default:	1
Required:	no

Description: Optional parameter.

Specifies the number of worker threads (at least 1) that syslog-ng PE uses to receive messages from the Google Pub/Sub messaging service. Increasing the number of worker threads can drastically improve the performance of the destination.

Preventing message duplication resulting from the At-Least-Once delivery behavior

The `google-pubsub()` source utilizes the At-Least-Once delivery behavior of Google Pub/Sub. This behavior is intentional, and its purpose is to avoid potential log loss. However, in certain cases, the At-Least-Once delivery behavior results in message duplication on the syslog-ng PE side.

Issue

On the Google Cloud Platform side, you can set the value of the **Acknowledgment deadline** (the default value is `10` seconds) when creating your Google Pub/Sub Subscription.

For more information, see Set up your Google Cloud project and Pub/Sub topic and subscriptions.

The syslog-ng PE application has to acknowledge `log-fetch-limit()` number of messages within the **Acknowledgement deadline** time limit. If syslog-ng PE does not acknowledge Google Pub/Sub messages no later than the time limit specified in the **Acknowledgment deadline**, the Google Pub/Sub service will attempt to redeliver the message to syslog-ng PE.

As a result, any acknowledgement sent later than the **Acknowledgment deadline** will result in message duplication on the syslog-ng PE side. This issue occurs most often if you have flow-control turned on, and your syslog-ng PE destinations are slow.

Workaround

To avoid message duplication, you can use one of these methods:

- Using the disk-buffer option if flow control is on

  When using the disk-buffer option, syslog-ng PE acknowledges Pub/Sub messages as soon as they are sent to the output queue or overflow queue, instead of acknowledging them when the destination sends or rewrites them.

- Adjusting the value of your `ack-tracker-timeout()` to the **Acknowledgment deadline**, and the value of your `ack-tracker-batch-size()` to your `log-fetch-limit()`

  The syslog-ng PE application acknowledges messages in batches. You can set the size (`ack-tracker-batch-size()`, the default value is `100`), and timeout (`ack-tracker-timeout()`, the default value is `3000` milliseconds, or 3 seconds) of these batches.

  To avoid message duplication, set your `ack-tracker-timeout()` to a value not larger than the value of your **Acknowledgment deadline**, and your `ack-tracker-batch-size()` to a value not larger than your `log-fetch-limit()`.

Error messages you may encounter while using the google-pubsub() source

Common error messages with workaround solutions

The following table describes the possible error messages that you may encounter while using the `google-pubsub()` source.

Error message	Possible reason(s)/resolution(s)
`DefaultCredentialsError: Could not automatically determine credentials. Please set GOOGLE_APPLICATION_CREDENTIALS or explicitly create credentials and re-run the application.`	Use the `credentials()` option to specify a key file. For more information, see Create service account credentials.
`404 Resource not found (resource=...).`	The specified subscription is not valid. Check the value of the `subscription()` option. Make sure you specify the Subscription ID, not the name of the subscription. For more information, see Set up your Google Cloud project and Pub/Sub topic and subscriptions.
`400 Invalid resource name given (name=...).`	The specified subscription is not valid. Check the value of the `subscription()` option. Make sure you specify the Subscription ID, not the name of the subscription.

Error message	Possible reason(s)/resolution(s)
	For more information, see Set up your Google Cloud project and Pub/Sub topic and subscriptions.
`404 Requested project not found or user does not have access to it (project=nonproj). Make sure to specify the unique project identifier and not the Google Cloud Console display name.`	Make sure to specify the unique project identifier and not the Google Cloud Console display name.
`PubSubSource init, failed to load credentials file, path=...`	The file specified in the `credentials()` option does not exist, or it is not accessible. Check the path and its permissions.

Error messages that require assistance from our Support Team

In some cases, you may encounter different error messages that require assistance from our Support Team. If you encounter similar error messages as those listed in the following table, One Identity recommends that you contact our Support Team for assistance.

Error message	Possible reason(s)/resolution(s)
`PubSubSource error while pulling messages, error: ...`	A non-retryable error occurred. If you encounter an error message other than those listed in the previous table, contact our Support Team with the complete error message.
`PubSubSource error while acking messages, error: ...`	A non-retryable error occurred. If you encounter an error message other than those listed in the previous table, contact our Support Team with the complete error message.

`wildcard-file`: Collecting messages from multiple text files

The `wildcard-file()` source collects log messages from multiple plain-text files from multiple directories. The `wildcard-file()` source is available in syslog-ng PE version 3.107.0.3 and later.

The syslog-ng PE 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 PE 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.

Declaration

```
wildcard-file(
       base-dir("<pathname>")
       filename-pattern("<filename>")
 );
```

Note the following important points:

- You can use the `*` and `?` wildcard characters in the filename (the `filename-pattern()` option), but not in the path (the `base-dir()` option).
- When using the `wildcard-file()` source, always set how often syslog-ng PE should check the files for new messages using the `follow-freq()` parameter.
- If you use multiple `wildcard-file()` sources in your configuration, 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.
- When using wildcards, syslog-ng PE monitors every matching file (up to the limit set in the `max-files()` option), 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. Set the `max-files()` option at least to the number of files you want to monitor. If the wildcard-file source matches more files than the value of the `max-files()` option, it is random which files will syslog-ng PE actually monitor. The default value of `max-files()` is 100.
- If the message does not have a proper syslog header, syslog-ng PE 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.

Required parameters: `base-dir()`, `filename-pattern()`. For the list of available optional parameters, see wildcard-file() source options.

Example: Using the wildcard-file() driver

The following example monitors every file with the `.log` extension in the `/var/log` directory for log messages.

wildcard-file() source options

The `wildcard-file()` driver has the following options:

base-dir()

Type:	path without filename
Default:	

Description: The path to the directory that contains the log files to monitor, for example, `base-dir("/var/log")`. To monitor also the subdirectories of the base directory, use the `recursive(yes)` option. For details, see recursive().

CAUTION:

If you use multiple `wildcard-file()` sources in your configuration, 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.

```
source  s_files  {
       wildcard-file(
             base-dir("/var/log")
             filename-pattern("*.log")
             recursive(no)
             follow-freq(1)
       );
 };
```

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 Message size and encoding.

filename-pattern()

Type:	filename without path
Default:	

Description: The filename to read messages from, without the path. You can use the `*` and `?` wildcard characters, without regular expression and character range support. You cannot use the `*` and `?` literally in the pattern.

For example, `filename-pattern("*.log")` matches the `syslog.log` and `auth.log` files, but does not match the `access_log` file. The `filename-pattern("*log")` pattern matches all three.

- `*`

  matches an arbitrary string, including an empty string

- `?`

  matches an arbitrary character

CAUTION:

If you use multiple `wildcard-file()` sources in your configuration, 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.

```
source  s_files  {
       wildcard-file(
             base-dir("/var/log")
             filename-pattern("*.log")
             recursive(no)
             follow-freq(1)
       );
 };
```

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-

	hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.

- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow-freq()

Type:	number
Default:	1

Description: Indicates that the source should be checked periodically. 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

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

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
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:	10000

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()*max-files()`. Note that to avoid performance problems, if `log-iw-size()/max-files()` is smaller than 100, syslog-ng PE automatically sets `log-iw-size()` to `max-files()*100`.

Example: Initial window size of file sources

If `log-fetch-limit()` is 100, and your wildcard file source has 200 files, then `log-iw-size()` should be at least 20000.

log-msg-size()

Type:	number (bytes)
Default:	Use the global `log-msg-size()` option, which defaults to `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program-override()` instead.

max-files()

Type:	integer
Default:	100

Description: Limits the number of files that the wildcard-file source monitors.

When using wildcards, syslog-ng PE monitors every matching file (up to the limit set in the `max-files()` option), 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. Set the `max-files()` option at least to the number of files you want to monitor. If the wildcard-file source matches more files than the value of the `max-files()` option, it is random which files will syslog-ng PE actually monitor. The default value of `max-files()` is 100.

monitor-method()

Type:	auto | inotify | poll
Default:	auto

Description: If the platform supports inotify, syslog-ng PE uses it automatically to detect changes to the source files. If inotify is not available, syslog-ng PE polls the files as set in the `follow-freq()` option. To force syslog-ng PE poll the files even if inotify is available, set this option to `poll`.

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.

To use the `multi-line-garbage()` option, set the `multi-line-mode()` option to `prefix-garbage`.

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()`.

multi-line-mode()

Type:	indented|regexp
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The syslog-ng PE application provides the following methods to process multi-line messages: `multi-line-mode(indented)`, and `multi-line-mode(prefix-garbage)`.

- The *indented* mode can process messages where each line that belongs to the previous line is indented by whitespace, and the message continues until the first non-indented line. For example, the Linux kernel (starting with version 3.5) uses this format for `/dev/log`, as well as several applications, like Apache Tomcat.

Example: Processing indented multi-line messages

```
source  s_tomcat  {
       file("/var/log/tomcat/xxx.log"
             multi-line-mode(indented)
       );
 };
```

- The *prefix-garbage* mode uses a string or regular expression (set in `multi-line-prefix()`) that matches the beginning of the log messages, 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. For details on using `multi-line-mode(prefix-garbage)`, see the `multi-line-prefix()` and `multi-line-garbage()` options.
- The `prefix-suffix` mode uses a string or regular expression (set in `multi-line-prefix()`) that matches the beginning of the log messages, ignores newline characters from the source until a line matches the regular expression set in `multi-line-suffix()`, and treats the lines between `multi-line-prefix()` and `multi-line-suffix()` as a single message. Any other lines between the end of the message and the beginning of a new message (that is, a line that matches the `multi-line-prefix()` expression) are discarded. For details on using `multi-line-mode(prefix-suffix)`, see the `multi-line-prefix()` and `multi-line-suffix()` options.

  The `prefix-suffix` mode is similar to the `prefix-garbage` mode, but it appends the garbage part to the message instead of discarding it.

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 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.

multi-line-prefix()

Type:	regular expression starting with the ^ character
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 (always start with the ^ character). 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.

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 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: 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-mode(regexp) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]
{2}\.") flags(no-parse));};
 };
```

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

multi-line-suffix()

Type:	regular expression
Default:	empty string

Description: Use the `multi-line-suffix()` option when processing multi-line messages. Specify a string or regular expression that matches the end of the multi-line message.

To use the `multi-line-suffix()` option, set the `multi-line-mode()` option to `prefix-suffix`. See also the `multi-line-prefix()` option.

multi-line-timeout()

Type:	number [seconds]
Default:	`[0 == disable]`

Description: If any `multi-line` option is enabled, there are two possible scenarios:

- the requirement specified in any `multi-line` option is met, which results in syslog-ng PE sending the full message (even multi-line messages)
- the number of seconds specified in `multi-line-timeout()` passes, which results in syslog-ng PE sending the current message, which could be partial

NOTE: One Identity recommends that you set `multi-line-timeout()` to a higher value than (preferably a multiple of) `follow-freq()`.

The recommended minimum value is 10 seconds.

pad-size()

Type:	number
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. 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`.

recursive()

Type:	yes | no
Default:	no

Description: When enabled, syslog-ng PE monitors every subdirectory of the path set in the `base-dir()` option, and reads log messages from files with matching filenames. The `recursive` option can be used together with wildcards in the filename.

CAUTION:

If you use multiple `wildcard-file()` sources in your configuration, 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.

Example: Monitoring multiple directories

The following example reads files having the `.log` extension from the `/var/log/` directory and its subdirectories, including for example, the `/var/log/apt/history.log` file.

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

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.

`linux-audit`: Collecting messages from Linux audit logs

This source reads and automatically parses the Linux audit logs. You can override the file name using the filename() parameter and the prefix for the created name-value pairs using the prefix() parameter. Any additional parameters are passed to the underlying file source.

NOTE: Most recent Linux distributions enable Security-Enhanced Linux (SELinux) or AppArmor as a security measure. If enabled, these technologies might disable access to the Linux Audit log file by default. Consult their manuals to enable Linux Audit log access for syslog-ng PE.

Declaration

```
linux-audit(options);
```

Example: Using the linux-audit() driver

```
source s_auditd {
    linux-audit(
        prefix("test.")
        hook-commands(
            startup("auditctl -w /etc/ -p wa")
```

```
            shutdown("auditctl -W /etc/ -p wa")
        )
    );
};
```

linux-audit() source options

The `linux-audit()` driver has the following options:

filename()

Type:	path
Default:	

Description: The log file of `linux-audit`. The syslog-ng PE application reads the Linux audit logs from this file.

prefix()

Synopsis:	prefix()
Default:	.auditd.

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. Note that if you use an empty prefix (`prefix("")`) or one starting with a dot, syslog-ng PE might replace the original value of an existing macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

mssql, oracle, sql: collecting messages from an SQL database

From version 7.0.21, syslog-ng Premium Edition (syslog-ng PE) can collect messages from a Microsoft SQL database (using an `mssql()` source), or from an Oracle database (using an `oracle()` source). For backward compatibility with syslog-ng PE 6LTS, `sql()` source is also available.

NOTE: For new deployments, One Identity recommends using a dedicated `mssql()` or `oracle()` source instead of a generic `sql()` source.

Currently, the Microsoft SQL (MSSQL) and Oracle databases are supported.

For more information, see Upgrading the sql() source of syslog-ng PE.

Limitations

- The Oracle database is not supported on Oracle Linux 6 and RedHat 6.
- The `sql()` source driver does not monitor rotated tables. As a result, every source can follow only one table.
- Timestamps with timezone are not supported. The syslog-ng PE application will retrieve the timestamps 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 columns as a complex ID.

Supported platforms

- `mssql()` source

  All Supported platforms.
- `oracle()` source

  All Supported platforms, except for Oracle Linux 6 and RedHat 6.

Declaration

The following examples can be used as a template for declaring the `mssql()`, `oracle()`, or `sql()` sources in your configuration.

Example: declaration for the `mssql()` source

You can use the following example as a template for declaring the `mssql()` source in your configuration:

```
source s_mssql {
    mssql(
        host(host)
        username(username)
        password(pass)
        port(1433)
        database(test_db)
        table(table_name)
        uid_column(uid)
        datetime_column(datetime)
        message_template('uid: ${.sql.uid} datetime: ${.sql.datetime}')
    );
};
```

Example: declaration for the `oracle()` source

You can use the following example as a template for declaring the `oracle()` source in your configuration:

```
source s_oracle {
    oracle(
        host(host)
        username(username)
        password(pass)
        port(1433)
        database(test_db)
        table(table_name)
        uid_column(uid)
        datetime_column(datetime)
        message_template('uid: ${.sql.uid} datetime: ${.sql.datetime}')
    );
};
```

Example: declaration for the `sql()` source

You can use the following example as a template for declaring the `sql()` source in your configuration:

```
source s_sql {
    sql(
       type(mssql)
       host(host)
       username(username)
       password(pass)
       port(1433)
       database(test_db)
       table(table_name)
       uid_column(uid)
       datetime_column(datetime)
       message_template('uid: ${.sql.uid} datetime: ${.sql.datetime}')
    );
};
```

Required parameters

The `mssql()`, `oracle()`, and `sql()` drivers have the following required parameters in common:

- `database()`
- `datetime-column()` or `date-column()` and `time-column()`
- `host()`
- `table()`
- `type()`
- `uid-column()`

The `oracle()` driver has an additional required parameter: `start-uid()`.

The `sql()` driver has an additional required parameter: `type()`.

mssql(), oracle(), and sql() source options

The `mssql()`, `oracle()`, and `sql()` drivers have the following options.

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

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

connection-timeout()

Type:	nonnegative integer
Default:	5

Description: Optional. The syslog-ng PE application waits `connection-timeout()` seconds for any request to complete. If a request does not succeed in the given time, syslog-ng PE will disconnect, and tries to reconnect after `time-reopen` seconds. `0` value means infinite timeout.

connect-query()

Type:	string
Default:	

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

CAUTION:

Hazard of data loss!

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 on your own responsibility.

Example: sample `connect-query()` statement

The following example is a sample `connect-query()` statement:

```
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.

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 strptime(3) - Linux man page.

datetime-column(col_name, [format])

Type:	string
Default:	

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 strptime(3) - Linux man page.

The following column types are supported:

- `oracle()`: `timestamp`, `int`
- `mssql()`: `datetime`, `int`

Example: sample `datetime-column(col_name, [format])`

The following example is a sample `datetime-column(col_name, [format])` column containing the timestamp:

```
columns("id","date","message")datetime("timestampcol", "%Y-%m-%d")
```

default-facility()

Type:	facility string
Default:	user

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

default-severity()

Type:	severity string
Default:	notice

Description: This parameter assigns a severity 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 PE

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 Customizing mssql() queries.

CAUTION:

Hazard of data loss!

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 on your own responsibility.

Example: sample `fetch-query()` statement
The following example is a sample `fetch-query()` statement:

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

The default `fetch-query()` statements for the `mssql()` and `oracle()` sources are the following:

- `mssql()`:

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

- `oracle()`:

  ```
  SELECT $columns FROM (SELECT * FROM $table WHERE $uid > $last_read_
  uid ORDER BY $uid) WHERE rownum <= $fetch_limit
  ```

follow-freq()

Type:	number(seconds)
Default:	If time-reopen() is set to a value other than the default `60`, the value of time-reopen(). Otherwise `60`.

Description: The syslog-ng PE application 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.

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.

log-fetch-limit()

Type:	
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop.

log-iw-size()

Type:	number (messages)
Default:	100

Description: The size of the initial window, this value is used during flow control.

login-timeout()

Type:	nonnegative integer
Default:	5

Description: Optional. The syslog-ng PE application waits `login-timeout()` seconds for the login request to complete. If the request does not succeed in the given time, syslog-ng PE will try to reconnect after `time-reopen()` seconds. `0` value means infinite timeout.

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.

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, 1521 for Oracle

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: sample `prefix()` - database column name - macro name correlation

The `prefix()` and database column name affect the name of the macro created from the database column.

For example, 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.

start-uid()

Type:	string
Default:	

Description:

- `mssql()`: Optional.
- `oracle()`: Mandatory. If `start-uid()` is specified, syslog-ng PE will only query entries with a strictly larger `uid`. After the first successfully fetched message, syslog-ng PE does not use this option anymore, even after restart or reload. If `start-uid()` is not specified, syslog-ng PE will read only the new records.

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 MSSQL queries, see Customizing mssql() queries.

CAUTION:

Hazard of data loss!

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 on your own responsibility.

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be enclosed between double quotes. When adding multiple tags, separate them with commas, for example, `tags("dmz", "router")`.

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 the `<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: sample queries' results for a table with two columns using the `template()` option

The following examples show the results of the first, and second queries when 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"'
```

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.

time-reopen()

Type:	number (seconds)
Default:	60

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

tls()

Type:	tls options
Default:	none

Description: The `tls()` configuration block enables TLS encryption for MSSQL Server.

NOTE: The `tls()` configuration block used for the `network()` source is similar in functionality, but there is no 100% feature parity between them. Consider the details for each `tls()` configuration block option before using them with your `mssql()` source.

The `tls()` configuration block has the following options:

- **ca-file()**

  Type: Filename

  Default: none

  Description: The name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the MSSQL server.

- **peer-verify()**

  Type: yes | no

  Default: yes

  Description: In syslog-ng PE version 7.0.28, this option only enables an extra check for the server CN matching against the hostname (configured in syslog-ng PE via the host() option).

  NOTE: This option is only effective if you also specify a ca-file() in your configuration block.

- **ssl-options()**

  Type: comma-separated list

  Possible values: no-tlsv1 | none

  Default: none

  Description: This option allows disabling TLS protocols. Consider that it only disables the current version, but not older versions.

  NOTE: The default none value does not restrict TLS protocols, but uses the default TLS value set for the mssql() source (1.0, 1.2).

type()

Type:	mssql, or oracle
Default:	

Description: Specifies the type of the database. Only use it with the generic sql() source.

uid-column()

Type:	string or string type(string) Supported column types are: integer, character type, datetime, datetime2
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.

The second optional parameter is a type hint (for example, `uid-column(uid' type ('DATE'))`. If provided, syslog-ng PE uses the given string in the SQL queries.

username()

Type:	string
Default:	n/a

Description: Name of the database user.

use-syslogng-pid()

Type:	yes | 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 PE process.

use-tnsnames()

Type:	yes | no
Default:	For `sql()`: yes For `oracle()`: no

Description: Optional. This option is used only in case of `oracle()`, or `sql()` source with `type("oracle")` option. If set to `yes`, syslog-ng PE will use the connection parameters defined in the `tnsnames.ora` configuration file.

Customizing mssql() queries

Every query executed by the `mssql()` 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:

Hazard of data loss!

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 on your own responsibility.

The available variables are the following:

`$last_read_uid`: The `uid` value of the last read record.

- `$columns`: Reach the `columns()` option of the MSSQL source (the default is `*`). If `$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"`).
- `$fetch_limit`: Reach the value of the `log-fetch-limit()` option.
- `$log_fetch_limit`: Alias for `$fetch_limit`.
- `$table`: The name of the `table()` option.
- `$uid`: The name of the `uid-column()`.
- `$uid_column`: Alias for `$uid`.

NOTE: The variables in the statement are not macro references.

Example: `mssql()` source `fetch-query()`

The following `mssql()` source `fetch-query()` example queries records that are older than the last record:

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

Configuring TLS encryption for MSSQL servers

From version 7.0.28, syslog-ng Premium Edition (syslog-ng PE) supports TLS encryption for MSSQL servers.

For more information, see Enable encrypted connections to the Database Engine in the *Microsoft SQL Docs* online documentation.

Prerequisites

Using TLS encryption for the MSSQL server with syslog-ng PE requires CA certificates correctly configured on the MSSQL server side.

Limitations

Using TLS encryption for the MSSQL server with syslog-ng PE has the following limitations:

NOTE: The `tls()` configuration block used for the `network()` source is similar in functionality, but there is no 100% feature parity between them. Consider the details for each `tls()` configuration block option before using them with your `mssql()` source.

Configuration

You can enable TLS encryption for your `mssql()` source on the syslog-ng PE side by including it in your configuration as follows:

```
source{
    mssql(
       host(...)
       ...
       tls(  ca-file(path/to/ca-file.pem)  peer-verify(yes)  ssl-options(no-tlsv1)
)
    );
 };
```

For further details about TLS encryption for your `mssql()` source, see the description of the `tls()` configuration block.

Enabling TLS encryption for your MSSQL server

To enable TLS encryption for MSSQL Server,

1. Open the **SQL Server Configuration Manager** (mmc snapin), navigate to **SQL Server Network Configuration > Protocols for SQL Server**, and right click **Protocols for SQL Server** to open the **Properties** page.
2. On the **Flags** tab, set **Force Encryption** to **yes**.
3. On the **Certificate** tab, use **TLS CA Certificate** to browse or import custom TLS CA Certificates for your encrypted connections.
4. To confirm your settings, click **OK**.
5. Under **SQL Server Services**, then restart **SQL Server instance**.

Possible connection errors between the MSSQL server (2019) and syslog-ng PE 7 LTS

While using MSSQL servers with syslog-ng Premium Edition (syslog-ng PE), you may run into connection errors.

This section provides information about possible connection errors, the return errors you encounter in your syslog-ng PE console log in these cases, the possible reasons behind the issues, and their possible resolutions.

For details about configuring TLS encryption for your MSSQL server, see Configuring TLS encryption for MSSQL servers.

Connection errors listed in this section:

- TCP/IP connection error between syslog-ng PE and the MSSQL Server
- Incorrect SQL server settings for allowed connections and user settings
- Login may not have proper permissions to the SQL server
- Low level logging with the mssql() source

TCP/IP connection error between syslog-ng PE and the MSSQL server

Console return error:

```
[2021-10-06T09:02:13.708984] Failed to connect to DB; connection-string='DRIVER=libtdsodbc.so;SERVER=mssql-server;PORT=1433;UID=sa;PWD=password;DATABASE=master;ClientCharset=UTF-8', ODBC-Diag='SQLRETURN: SQL_ERROR(-1) | entry: 1, SQLSTATE: 08S01, error_code: 20009, error_string: [FreeTDS][SQL Server]Unable to connect: Adaptive Server is unavailable or does not exist | entry: 2, SQLSTATE: 08001, error_code: 0, error_string: [FreeTDS][SQL Server]Unable to connect to data source', driver='s_mssql_win_2019#0'
```

Possible reason: There is a TCP/IP connection error between syslog-ng PE and the MSSQL server.

Possible resolution: Enable TCP/IP protocol over the SQL server.

To enable TCP/IP protocol over SQL server,

1. Open the SQL Server Configuration Manager (mmc snapin), and navigate to **SQL Server Network Configuration > Protocols for SQL Server**.
2. Next to **TCP/IP**, double click on the **Disabled** status.
3. Next to **Enable**, change the option from **No** to **Yes**.
4. Switch to the **IP Addresses** tab.
5. In the **IPAll** settings node list, find **TCP port**, and set it to **1433**. .
6. To confirm the settings, click **OK**.
7. Under **SQL Server Services**, select the SQL server instance you want to restart.
8. Check your syslog-ng PE connection again.

Incorrect SQL server settings for allowed connections and user settings

Console return error:

[2021-10-06T09:16:04.719213] Failed to connect to DB; connection-string='DRIVER=libtdsodbc.so;SERVER=mssql-server;PORT=1433;UID=test_user;PWD=password;DATABASE=master;ClientCharset=UTF-8', ODBC-Diag='SQLRETURN: SQL_ERROR(-1) | entry: 1, SQLSTATE: 42000, error_code: 18456, error_string: [FreeTDS][SQL Server]Login failed for user \'test_user\'. | entry: 2, SQLSTATE: 08001, error_code: 0, error_string: [FreeTDS][SQL Server]Unable to connect to data source', driver='s_mssql_win_2019#0'

Possible reason: Your SQL server settings for allowed connections and user settings are incorrect.

Possible resolutions:

To resolve the issue, try one of the following methods:

- **Check Server Authentication mode in your MSSQL server settings**
  1. Start Microsoft SQL Server Management Studio and log in with the following credentials:
     - Server type: **Database Engine**
     - Server name: `<your-current-installation>` (default)
     - Authentication:**Windows Authentication**
  2. Open **Server Properties** and navigate to **Security page**.
  3. Under **Server Authentication**, enable the following **SQL Server and Windows Authentication mode**.
  4. Open the SQL Server Configuration Manager (mmc snapin), and navigate to **SQL Server Services, Restart SQL Server instance**.
- **Check Remote server connections in your MSSQL server settings**
  1. Log in to Microsoft SQL Server Management Studio with the previously used mode.
  2. Open **Server Properties** and navigate to the **Connections** page.
  3. Under **Remote server connections**, check if **Allow remote connections to this server** is enabled.
- **Check SQL Login status and Roles**
  1. Log in to Microsoft SQL Server Management Studio with the previously used mode.
  2. Navigate to **Security > Logins**, and right click **Login** to open the **Properties** page for your Login user.
  3. Check that all of the following pages have the right settings for your Login user:
     - **General**
     - **Server Roles**
     - **User mapping**

- **Securables**
- **Status**

Login may not have proper permissions to the SQL server

Console return error:

```
[2021-10-06T09:22:51.479930] Failed to execute fetch-query; ODBC-
Diag='SQLRETURN: SQL_ERROR(-1) | entry: 1, SQLSTATE: 42000, error_code: 297,
error_string: [FreeTDS][SQL Server]The user does not have permission to perform
this action. | entry: 2, SQLSTATE: 42000, error_code: 300, error_string:
[FreeTDS][SQL Server]VIEW SERVER STATE permission was denied on object
\'server\', database \'master\'.', driver='s_mssql_win_2019#0'
```

Possible reason: Login may not have proper permissions to the SQL server.

Possible resolutions:

To resolve the issue, try one of the following methods check SQL Login status and Roles:

1. Log in to Microsoft SQL Server Management Studio with the previously used mode.
2. Navigate to **Security > Logins**, and right click **Login** to open the **Properties** page for your Login user.
3. Check that the **Server Roles** page has the right status and roles for your Login user.

Low level logging with the mssql() source

In case of the `mssql()` source, the user can enable an extra level of logging. Developers and users new to syslog-ng PE commonly use this option.

You can use the following debug logging methods:

- **The syslog-ng PE PE7 FreeTDS verbose debug log method**

  If you start syslog-ng PE with the following enabled environment variable, you can find FreeTDS debug logs in the configured output file:

  ```
  export TDSDUMP=/tmp/freetds_debug.log
  ```

  TIP: This debug logging method is recommended if you start syslog-ng PE from service.

- **The syslog-ng PE PE7 FreeTDS stdout debug log method**

  If you start syslog-ng PE with the following enabled environment variable, you can find FreeTDS debug logs in syslog-ng PE's `stdout`:

  ```
  export TDSDUMP=stdout
  ```

TIP: This debug logging method is recommended if you start syslog-ng PE from the foreground.

network: 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 ALTP protocol as well. For details about the ALTP protocol, see Advanced 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.
- 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 Encrypting log messages with TLS.

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/514 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: Using the network() driver

Using only the default settings: listen on every available IPV4 interface on the TCP/514 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 syslog: 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 network() source options.

NOTE: To receive UDP messages at very high message rate, you can use the `udp-balancer()` source. For details, see udp-balancer: Receiving UDP messages at very high rate.

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 Message size and encoding.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.

- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.

- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]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 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()

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

Type:	number
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 `ip-protocol(4)`.

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)`. You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source  s_network_tcp  {
       network(
             transport("tcp")
             ip("::")
             ip-protocol(6)
             port(601)
       );
 };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to sources should be closed when syslog-ng is 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 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
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:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with 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: 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 `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections.

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. 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
Default:	In case of TCP transport: 514 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
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 this 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
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:	altp, udp, tcp, tls, proxied_tcp, or proxied_tls
Default:	tcp

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

For detailed information about how from version 7.0.23 onwards, syslog-ng PE supports the proxied_tcp parameter and the proxied_tls parameter, see Proxy Protocol support.

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 this 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.

trim-large-messages()

Type:	yes|no
Default:	Use the global trim-large-messages() option, which defaults to no.

Description: Determines what syslog-ng PE does with incoming log messages that are received using the IETF-syslog protocol using the syslog() driver, and are longer than the value of log-msg-size(). Other drivers ignore the trim-large-messages() option.

- If set to no, syslog-ng PE drops the incoming log message.
- If set to yes, syslog-ng PE trims the incoming log message to the size set in log-msg-size(), and adds the trimmed tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

```
filter  f_trimmed  {
        tags("trimmed");
 };
```

If syslog-ng PE trims a log message, it sends a debug-level log message to its `internal()` source.

As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in syslog-ng PE version 7.0.143.21 and later.

Uses the value of the global option if not specified.

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 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: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

TIP: Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.

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 | 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 PE process.

Proxy Protocol support

If you connect load balancers to your syslog-ng PE application, syslog-ng PE identifies every connection that is connected to the load balancers identically by default, regardless of the source IP or the source protocol. Essentially, the load balancer masks the source IP unless you enable Proxy Protocol support for your proxy TLS `transport()` to inject information about the original connection into the forwarded TCP session.

For further details about the working mechanism behind the Proxy Protocol support on syslog-ng PE and the configuration details, see the following sections:

The working mechanism behind the Proxy Protocol

This section describes how syslog-ng Premium Edition (syslog-ng PE) supports the Proxy Protocol.

The working mechanism behind the Proxy Protocol

When using the Proxy Protocol during load balancing, syslog-ng PE detects the information behind connections connected to the load balancer, then parses the injected information and adds the following macros to every message the comes through the connection later on:

- `PROXY_SRCIP` (the source IP of the proxy)
- `PROXY_SRCPORT` (the source port of the proxy)
- `PROXY_DSTIP` (the destination IP of the proxy)
- `PROXY_DSTPORT` (the destination port of the proxy)

NOTE: Consider the following about macros and headers:

- When the proxy protocol header is `PROXY UNKNOWN`, no additional macros are added.
- When syslog-ng PE cannot parse a proxy protocol header, the connection is closed:

```
[2020-11-20T17:33:22.189458] PROXY protocol header received;
line='PROXYdsfj'
[2020-11-20T17:33:22.189475] Error parsing PROXY protocol header;
[2020-11-20T17:33:22.189517] Syslog connection closed; fd='13',
client='AF_INET(127.0.0.1:51665)', local='AF_INET(0.0.0.0:6666)'
[2020-11-20T17:33:22.189546] Freeing PROXY protocol source driver;
driver='0x7fffcba5bcf0'
[2020-11-20T17:33:22.189600] Closing log transport fd; fd='13'
```

NOTE: Since the driver only implements version 1 of the protocol, it only supports TCP4 and TCP6 connections. TLS connections also supported.

Proxy Protocol: configuration and output examples

This section provides information about enabling Proxy Protocol support in your `network()` source options, and an example configuration and output to illustrate how the Proxy Protocol method works in syslog-ng Premium Edition (syslog-ng PE).

For more information about the working mechanism of the Proxy Protocol, see The working mechanism behind the Proxy Protocol.

Enabling Proxy Protocol support for your network() source options

Unless you enable Proxy Protocol support for your `network()` source, syslog-ng PE identifies every connection that is connected to the load balancers identically by default, regardless of the source IP or the source protocol.

To enable Proxy Protocol for your `network()` source, set the `transport()` parameter of your `network()` source to `proxied_tcp` or `proxied_tls`, depending on your preference and configuration.

When you enable Proxy Protocol support for your `network()` source, you can use the following configuration example with your syslog-ng PE application.

Configuration

The following code sample illustrates how you can use the Proxy Protocol in your syslog-ng PE configuration (using the `transport()` parameter set to `proxied_tls`).

```
@version: 3.30

source s_tcp_pp {
  network (
    port(6666)
    transport("proxied_tls")
    tls(
```

```
        key-file("/certs/certs/server/server.rsa")
        cert-file("/certs/certs/server/server.crt")
        ca-dir("/certs/certs/CA")
        peer-verify("required-trusted")
    )
  );
};

destination d_file {
  file("/var/log/proxy-proto.log" template("$(format-json --scope nv-
pairs)\n"));
};

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

With this configuration, the Proxy Protocol method will perform injecting the information of the original connection into the forwarded TCP session, based on the working mechanism described in The working mechanism behind the Proxy Protocol.

The following example illustrates how the parsed macros will appear in the output.

Example: Output for the `PROXY TCP4 192.168.1.1 10.10.0.1 1111 2222` input header

With the `PROXY TCP4 192.168.1.1 10.10.0.1 1111 2222` input header, the output looks like this:

```
{"SOURCE":"s_tcp_pp","PROXIED_SRCPORT":"1111","PROXIED_
SRCIP":"192.168.1.1","PROXIED_IP_VERSION":"4","PROXIED_
DSTPORT":"2222","PROXIED_
DSTIP":"10.10.0.1","PROGRAM":"TestMsg","MESSAGE":"","LEGACY_
MSGHDR":"TestMsg","HOST_FROM":"localhost","HOST":"localhost"}
```

Note that the macros that syslog-ng PE adds to the message appear in the output.

office365: Fetching logs from Office 365

Starting with syslog-ng PE version 7.0.17, you can fetch logs from your Office 365 account using the Office 365 Management Activity API.

The syslog-ng PE application supports every content type of the Management Activity API using a corresponding source driver:

- `Audit.AzureActiveDirectory`: `office365-azure-active-directory()`
- `Audit.Exchange`: `office365-exchange()`
- `Audit.General`: `office365-general()`
- `Audit.SharePoint`: `office365-sharepoint()`
- `DLP.All`: `office365-dlp()`

Limitations

- In some cases, the logs will appear only 24-48 hours after successfully configuring syslog-ng PE and Office 365.
- Due to the distributed nature of the Office 365 log management architecture, there is a synchronization interval in the Office 365 Management Activity API. During this interval, the messages returned to queries can be inconsistent. To avoid this synchronization window, syslog-ng PE does not fetch the logs in real-time, only 15 minutes after the message becomes available in the management API. This means that there is a 15-minute latency between the logs available in the Office 365 Management Activity API and syslog-ng PE.

Declaration

```
source  s_office365  {
       office365-<content-type>(
             tenant_id('tenant-id')
             client_id('client-id')
             client_secret('client-secret')
       );
 };
```

Example: Fetching Azure Active Directory logs from Office 365

The following example configuration fetches logs from and Audit.AzureActiveDirectory subscription using the `office365-azure-active-directory()` source driver.

```
@version: 7.0
@include "scl.conf"

source s_o365_ad {
    office365-azure-active-directory(
        tenant_id('tenant-id')
        client_id('client-id')
        client_secret('client-secret')
```

```
    );
};

destination d_file { file("/tmp/o365_ad_out.log"); };

log {
    source(s_o365_ad);
    destination(d_file);
    flags(flow-control);
};
```

Configuring Office 365 to permit fetching logs

This procedure summarizes how to configure Office 365 to permit syslog-ng Premium Edition (syslog-ng PE) to fetch log messages using the Office 365 Management Activity API.

Prerequisites

- A valid Office 365 account (for example: Azure Active Directory Premium P2).
- Administrator permissions to the Office 365 account.
- Content subscription must be enabled for the content types you want to fetch logs from.
- syslog-ng Premium Edition version 7.0.15 or later.

1. The syslog-ng PE application communicates with the Management Activity API through a service app registered in Azure. Complete the following steps to register a service app. For details on registering an app, see the Office 365 documentation.
    a. Login to https://portal.azure.com.
    b. Register a new web application (for example, `syslog-service-app`).
    c. Grant all **application permissions** and **delegated permissions** to the created app.
    d. Create secret (password) for the app. Record the password somewhere, as you will need it later to configure syslog-ng PE. If you set an expiration to the password, make sure to renew the password and update it in your syslog-ng PE configuration file before it expires.
    e. In the authentication settings of the new app, enable **Access Tokens**.
2. Enabling audit logging in Office 365. Note that your user must have an administrator permissions to enable audit logging. For details on enabling audit logging, see the

Office 365 documentation. If you have problems enabling audit logging, see Troubleshooting audit logging in Office 365

3. Start content subscriptions for the content types you want to fetch logs from using the following command. The `office365subscriptiontool` utility is automatically installed with syslog-ng PE version 7.0.15 and newer.

```
%syslog-ng-installation-directory%/bin/office365subscriptiontool start --
tenant-id=[your tenant-id] --client-id=[your client-id] --client-secret=
[your client-secret in escaped format] --content-type=[content-type-to-
fetch-logs-from]
```

Note that special characters (for example, `?/*+!$`) in the client secret must be escaped (according to the rules of escaping in bash/shell) when starting the content subscription.

In your syslog-ng PE configuration, you will have to use the source driver matching the content type of the content subscription. For example, use the `office365-azure-active-directory()` driver for the `Audit.AzureActiveDirectory` content type.

Repeat this step for every content type you want to fetch logs from.

> NOTE: When a subscription is created, it can take up to 12 hours for the first content blobs to become available for that subscription.

office365() source options

The `office365()` drivers have the following options. The `tenant-id()`, `client-id()`, and `client-secret()` option is required, the others are optional.

client-id()

Type:	string
Default:	

Description: The ID of the service app. For details, see Configuring Office 365 to permit fetching logs.

client-secret()

Type:	string
Default:	

Description: The password of the service app. For details, see Configuring Office 365 to permit fetching logs.

If you set an expiration to the password, make sure to renew the password and update it in your syslog-ng PE configuration file before it expires.

fetch-no-data-delay()

Type:	real number [seconds]
Default:	The value of `time-reopen()`

Description: When there are no messages available in a poll event, the Office 365 Management Activity API waits `fetch-no-data-delay()` seconds before the next poll.

NOTE: You can set the `fetch-no-data-delay()` value to fractions, for example:

```
fetch-no-data-delay(0.5)
```

persist-name()

Type:	string
Default:	None

*Description:*If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

tenant-id()

Type:	string
Default:	

Description: The ID of the Tenant that the Office 365 account belongs to.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

By default, this option uses the value set in the `time-reopen()` global option.

Troubleshooting audit logging in Office 365

If you have trouble with enabling audit logging in Office 365, complete the following steps.

1. Login to https://protection.office.com/.
2. Verify that the **Search > Audit log search** context is active. If it is not, execute the following commands from PowerShell:

```
$ $UserCredential = Get-Credential
# use your Azure account
$ $Session = New-PSSession -ConfigurationName Microsoft.Exchange -
ConnectionUri https://outlook.office365.com/powershell-liveid/ -Credential
$UserCredential -Authentication Basic -AllowRedirection
$ Import-PSSession $Session -DisableNameChecking
$ Enable-OrganizationCustomization
# Wait 4 hours
$ Set-AdminAuditLogConfig -UnifiedAuditLogIngestionEnabled $true
# Wait 1 hour
$ Set-AdminAuditLogConfig -UnifiedAuditLogIngestionEnabled $false
# Wait 1 hour
$ Set-AdminAuditLogConfig -UnifiedAuditLogIngestionEnabled $true
# Wait 1 hour
```

3. Login to https://protection.office.com/ and check if there are audit logs under **Search > Audit log search**. Note that in some cases, the logs will appear only after 24-48 hours.

`osquery`: Collect and parse osquery result logs

The osquery application allows you to ask questions about your machine using an SQL-like language. For example, you can query running processes, logged in users, installed packages and syslog messages as well. You can make queries on demand, and also schedule them to run regularly.

The `osquery()` source of syslog-ng PE allows you read the results of periodical osquery queries (from the `/var/log/osquery/osqueryd.results.log` file) and automatically parse the messages (if you want to use syslog-ng PE to send log messages to osquery, read this blogpost). For example, you can:

- Create filters from the fields of the messages.
- Limit which fields to store, or create additional fields (combine multiple fields into one filed, and so on).

- Send the messages to a central location, for example, to Elasticsearch, directly from syslog-ng PE.

The syslog-ng PE application automatically adds the `.osquery.` prefix to the name of the fields the extracted from the message.

The `osquery()` source is available in syslog-ng PE version 3.107.0.4 and later.

Prerequisites

- To use the `osquery()` driver, the `scl.conf` file must be included in your syslog-ng PE configuration:

  ```
  @include "scl.conf"
  ```

- syslog-ng PE must be compiled with JSON-support enabled.

The `osquery()` driver is actually a reusable configuration snippet configured to read the osquery log file using the `file()` driver, and process its JSON contents. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of this configuration snippet on GitHub.

Example: Using the osquery() driver with the default settings

The following syslog-ng PE configuration sample uses the default settings of the driver, reading osquery result logs from the `/var/log/osquery/osqueryd.results.log` file, and writes the log messages generated from the traps into a file.

```
@version: 7.0
@include "scl.conf"
source s_osquery {
      osquery();
};
log {
      source(s_osquery);
      destination {
            file("/var/log/example.log");
      };
};
```

Filter for messages related to loading Linux kernel modules:

```
@version: 7.0
@include "scl.conf"
source s_osquery {
    osquery();
};
```

```
log {
    source(s_osquery);
    filter f_modules {
        "${.osquery.name}" eq "pack_incident-response_kernel_modules"
    };
    destination {
        file("/var/log/example.log");
    };
};
```

Example: Using the osquery() driver with custom configuration

The following syslog-ng PE configuration sample reads osquery result logs from the `/tmp/osquery_input.log` file, and writes the log messages generated from the traps into a file. Using the `format-json` template, the outgoing message will be a well-formed JSON message.

Input message

```
{"name":"pack_osquery-monitoring_osquery_
info","hostIdentifier":"testhost","calendarTime":"Fri Jul 21 10:04:41 2017
UTC","unixTime":"1500631481","decorations":{"host_uuid":"4C4C4544-004D-
3610-8043-C2C04F4D3332","username":"myuser"},"columns":{"build_
distro":"xenial","build_platform":"ubuntu","config_
hash":"43cd1c6a7d0c283e21e026a53e619b2e582e94ee","config_
valid":"1","counter":"4","extensions":"active","instance_id":"d0c3eb0d-
f8e0-4bea-868b-18a2c61b438d","pid":"19764","resident_
size":"26416000","start_time":"1500629552","system_time":"223","user_
time":"476","uuid":"4C4C4544-004D-3610-8043-
C2C04F4D3332","version":"2.5.0","watcher":"19762"},"action":"added"}
```

syslog-ng PE configuration

```
@version: 7.0
@include "scl.conf"
source s_osquery {
    osquery(
      file(/tmp/osquery_input.log)
      prefix(.osquery.)
```

```
    );
};
destination d_file {
    file("/tmp/output.txt"
      template("$(format_json --key .osquery.*)\n")
    );
};
log {
    source(s_osquery);
    destination(d_file);
    flags(flow-control);
};
```

Outgoing message

```
Outgoing message; message='{"_osquery":
{"unixTime":"1500631481","name":"pack_osquery-monitoring_osquery_
info","hostIdentifier":"testhost","decorations":
{"username":"myuser","host_uuid":"4C4C4544-004D-3610-8043-
C2C04F4D3332"},"columns":
{"watcher":"19762","version":"2.5.0","uuid":"4C4C4544-004D-3610-8043-
C2C04F4D3332","user_time":"476","system_time":"223","start_
time":"1500629552","resident_size":"26416000","pid":"19764","instance_
id":"d0c3eb0d-f8e0-4bea-868b-
18a2c61b438d","extensions":"active","counter":"4","config_
valid":"1","config_
hash":"43cd1c6a7d0c283e21e026a53e619b2e582e94ee","build_
platform":"ubuntu","build_distro":"xenial"},"calendarTime":"Fri Jul 21
10:04:41 2017 UTC","action":"added"}}\x0a'
```

To configure a destination to send the log messages to Elasticsearch, see elasticsearch2: Sending messages directly to Elasticsearch version 2.0 or higher (DEPRECATED). For other destinations, see Sending and storing log messages — destinations and destination drivers.

osquery() source options

The `osquery()` driver has the following options.

file()

Type:	path
Default:	/var/log/osquery/osqueryd.results.log

Description: The log file of `osquery` that stores the results of periodic queries. The syslog-ng PE application reads the messages from this file.

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

Default value

`.osquery.` option.

`pipe`: 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 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 flags().**

Example: 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, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.

- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow-freq()

Type:	number
Default:	1

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

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

Type:	number (bytes)
Default:	Use the global `log-msg-size()` option, which defaults to `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program-override()` instead.

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.

To use the `multi-line-garbage()` option, set the `multi-line-mode()` option to `prefix-garbage`.

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()`.

multi-line-mode()

Type:	indented|regexp
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The syslog-ng PE application provides the following methods to process multi-line messages: `multi-line-mode(indented)`, and `multi-line-mode(prefix-garbage)`.

- The *indented* mode can process messages where each line that belongs to the previous line is indented by whitespace, and the message continues until the first non-indented line. For example, the Linux kernel (starting with version 3.5) uses this format for `/dev/log`, as well as several applications, like Apache Tomcat.

  Example: Processing indented multi-line messages

  ```
  source  s_tomcat  {
        file("/var/log/tomcat/xxx.log"
              multi-line-mode(indented)
        );
  };
  ```

- The *prefix-garbage* mode uses a string or regular expression (set in `multi-line-prefix()`) that matches the beginning of the log messages, 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. For details on using `multi-line-mode(prefix-garbage)`, see the `multi-line-prefix()` and `multi-line-garbage()` options.

- The `prefix-suffix` mode uses a string or regular expression (set in `multi-line-prefix()`) that matches the beginning of the log messages, ignores newline characters from the source until a line matches the regular expression set in `multi-line-suffix()`, and treats the lines between `multi-line-prefix()` and `multi-line-suffix()` as a single message. Any other lines between the end of the message and the beginning of a new message (that is, a line that matches the `multi-line-prefix()` expression) are discarded. For details on using `multi-line-mode(prefix-suffix)`, see the `multi-line-prefix()` and `multi-line-suffix()` options.

  The `prefix-suffix` mode is similar to the `prefix-garbage` mode, but it appends the garbage part to the message instead of discarding it.

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 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.

multi-line-prefix()

Type:	regular expression starting with the ^ character
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 (always start with the ^ character). 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.

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 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: 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-mode(regexp) multi-line-prefix("[0-9]{4}\.[0-9]{2}\.[0-9]
{2}\.") flags(no-parse));};
 };
```

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

multi-line-suffix()

Type:	regular expression
Default:	empty string

Description: Use the `multi-line-suffix()` option when processing multi-line messages. Specify a string or regular expression that matches the end of the multi-line message.

To use the `multi-line-suffix()` option, set the `multi-line-mode()` option to `prefix-suffix`. See also the `multi-line-prefix()` option.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad-size()

Type:	number
Default:	0

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

program: 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: 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, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.

- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]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
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.

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

inherit-environment()

Type:	yes|no
Default:	yes

Description: By default, when `program()` starts an external application or script, it inherits the entire environment of the parent process (that is, syslog-ng PE). Use `inherit-environment(no)` to prevent this.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

Type:	number (bytes)
Default:	Use the global `log-msg-size()` option, which defaults to `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program-override()` instead.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad-size()

Type:	number
Default:	0

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

python: writing server-style Python sources

The Python source allows you to write your own source in Python.

You can write two different type of sources in Python:

- Server-style sources that receives messages. Write server-style sources if you want to use an event-loop based, nonblocking server framework in Python, or if you want to implement a custom loop.
- Fetcher-style sources that actively fetch messages. In general, write fetcher-style sources (for example, when using simple blocking APIs), unless you explicitly need a server-style source.

This section describes server-style sources. For details on fetcher-style sources, see python-fetcher: writing fetcher-style Python sources.

The following points apply to using Python blocks in syslog-ng PE in general:

- Only the default Python modules are available (that is, you cannot import external Python modules, and One Identity does not support using external Python modules).
- The syslog-ng PE application uses its own Python interpreter (shipped with the default syslog-ng PE installation) instead of the system's Python interpreter.
- The syslog-ng PE application is shipped with Python version 3.8.
- The Python block must be a top-level block in the syslog-ng PE configuration file.
- If you store the Python code in a separate Python file and only include it in the syslog-ng PE configuration file, make sure that the PYTHON_PATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start syslog-ng PE manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`

  In production, when syslog-ng PE starts on boot, you must configure your startup script to include the Python path. The exact method depends on your operating system. For recent Red Hat Enterprise Linux, Fedora, and CentOS distributions that use systemd, the `systemctl` command sources the `/etc/sysconfig/syslog-ng` file before starting syslog-ng PE. (On openSUSE and SLES, `/etc/sysconfig/syslog` file.) Append the following line to the end of this file: `PYTHONPATH="<path-to-your-python-file>"`, for example, `PYTHONPATH="/opt/syslog-ng/etc"`
- The Python object is initiated every time when syslog-ng PE is started or reloaded.

  CAUTION:

  If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.
- The Python block can contain multiple Python functions.
- Using Python code in syslog-ng PE can significantly decrease the performance of syslog-ng PE, especially if the Python code is slow. In general, the features of syslog-ng PE are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The syslog-ng PE application does not do any of this.
- Python error messages are available in the `internal()` source of syslog-ng PE.
- You can access the name-value pairs of syslog-ng PE directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of syslog-ng PE. For details, see Logging from your Python code.

- **Support disclaimer**

  CAUTION:

  This is a Preview Feature, which provides an insight to planned enhancements to functionality in the product. Consider this Preview Feature a work in progress, as it may not represent the final design and functionality.

  This feature has completed QA release testing, but its full impact on production systems has not been determined yet, and potential future changes in functionality and the user interface may result in compatibility issues in your current settings.

  One Identity recommends the following:

  - **Consider the potential risks when using this functionality in a production environment.**
  - **Consider the Support Policy on Product Preview Features before using this functionality in a production environment.**
  - **Closely and regularly keep track of official One Identity announcements about potential changes in functionality and the user interface. If these potential changes affect your configuration, check the changes you have to make in your configuration, otherwise your syslog-ng PE application may not start after upgrade.**
  - **Always perform tests prior to upgrades in order to avoid the risks mentioned.**

  However, you are welcome to try this feature and if you have any feedback, Contact One Identity.

  Support Policy on Product Preview Features

  The One Identity Support Team will:

  - **Accept and review each service request opened regarding a Preview Feature.**
  - **Consider all service requests relating to a Preview Features as severity level 3.**
  - **Provide best effort support to resolve any issues relating to a Preview Feature.**
  - **Work with customers to log any product defects or enhancements relating to Preview Features.**
  - **Not accept requests for escalations regarding Preview Features.**
  - **Not provide after-hours support for Preview Features.**

  Using Python in syslog-ng PE is recommended only if you are familiar with both Python and syslog-ng PE. One Identity is not responsible for the quality, resource requirements, or any bugs in the Python code, nor any syslog-ng PE crashes,

message losses, or any other damage caused by the improper use of this feature, unless explicitly stated in a contract with One Identity.

NOTE: Starting with 7.0.193.0.26, syslog-ng PE assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your syslog-ng PE configuration, add a unique `persist-name()` to each source or destination, otherwise syslog-ng PE will not start. For example:

```
log {
       source { python(class(PyNetworkSource) options("port" "8080")
persist-name("<unique-string>"); };
       source { python(class(PyNetworkSource) options("port" "8081")); };
    };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:

```
from syslogng import LogSource
  class PyNetworSource(LogSource):
    @staticmethod
    def generate_persist_name(options):
        return options["port"]
    def run(self):
        pass
    def request_exit(self):
        pass
```

Declaration

Python sources consist of two parts. The first is a syslog-ng PE source object that you define in your syslog-ng PE configuration and use in the log path. This object references a Python class, which is the second part of the Python source. The Python class receives or fetches the log messages, and can do virtually anything that you can code in Python. You can either embed the Python class into your syslog-ng PE configuration file, or store it in an external Python file.

```
source <name_of_the_python_source>{
    python(
        class("<name_of_the_python_class_executed_by_the_source>")
        options(
            "option1" "value1",
            "option2" "value2"
        )
    );
};

python {
```

```
from syslogng import LogSource
from syslogng import LogMessage

class <name_of_the_python_class_executed_by_the_source>(LogSource):
    def init(self, options): # optional
        print("init")
        print(options)
        self.exit = False
        return True

    def deinit(self): # optional
        print("deinit")

    def run(self): # mandatory
        print("run")
        while not self.exit:
            # Must create a message
            msg = LogMessage("this is a log message")
            self.post_message(msg)

    def request_exit(self): # mandatory
        print("exit")
        self.exit = True
};
```

Methods of the python() source

Server-style Python sources must be inherited from the `syslogng.LogSource` class, and must implement at least the `run` and `request_exit` methods. Multiple inheritance is allowed, but only for pure Python super classes.

You can implement your own event loop, or integrate the event loop of an external framework or library, for example, KafkaConsumer, Flask, Twisted engine, and so on.

To post messages, call `LogSource::post_message()` method in the `run` method.

`init(self, options)` method (optional)

The syslog-ng PE application initializes Python objects every time when it is started or reloaded. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your source to work.

CAUTION:

If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.

When this method returns with False, syslog-ng PE does not start. It can be used to check options and return False when they prevent the successful start of the source.

`options`: This optional argument contains the contents of the `options()` parameter of the syslog-ng PE configuration object as a Python dictionary.

`run(self)` method (mandatory)

Use the `run` method to implement an event loop, or start a server framework or library. Create `LogMessage` instances in this method, and pass them to the log paths by calling `LogSource::post_message()`.

Currently, `run` stops permanently if an unhandled exception happens.

For details on parsing and posting messages, see Python LogMessage API.

`request_exit(self)` method (mandatory)

The syslog-ng PE application calls this method when syslog-ng PE is shut down or restarted. The `request_exit` method must shut down the event loop or framework, so the `run` method can return gracefully. If you use blocking operations within the `run()` method, use `request_exit()` to interrupt those operations and set an exit flag, otherwise syslog-ng PE is not able to stop. Note that syslog-ng PE calls the `request_exit` method from a thread different from the source thread.

The `deinit(self)` method (optional)

This method is executed when syslog-ng PE is stopped or reloaded. This method does not return a value.

> **CAUTION:**
>
> **If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.**

For the list of available optional parameters, see python() and python-fetcher() source options.

Python LogMessage API

The LogMessage API allows you to create LogMessage objects in Python sources, parse syslog messages, and set the various fields of the log message.

`LogMessage()` method: Create log message objects

You can use the `LogMessage()` method to create a structured log message instance. For example:

```
from syslogng import LogMessage

msg = LogMessage() # Initialize an empty message with default values (recvd
timestamp, rcptid, hostid, ...)
msg = LogMessage("string or bytes-like object") # Initialize a message and set
its ${MESSAGE} field to the specified argument
```

You can also explicitly set the different values of the log message. For example:

```
msg["MESSAGE"] = "message"
msg["HOST"] = "hostname"
```

You can set certain special field (timestamp, priority) by using specific methods.

Note the following points when creating a log message:

- When setting the hostname, syslog-ng PE takes the following hostname-related options of the configuration into account: `chain-hostnames()`, `keep-hostname()`, `use-dns()`, and `use-fqdn()`.
- Python sources ignore the `log-msg-size()` option.
- The syslog-ng PE application accepts only one message from every `LogSource::post_message()` or `fetch()` call, batching is currently not supported. If your Python code accepts batches of messages, you must pass them to syslog-ng PE one-by-one. Similarly, if you need to split messages in the source, you must do so in your Python code, and pass the messages separately.
- Do not reuse or store LogMessage objects after posting (calling `post_message()`) or returning the message from `fetch()`.

`parse()` method: Parse syslog messages

The `parse()` method allows you to parse incoming messages as syslog messages. By default, the `parse()` method attempts to parse the message as an IETF-syslog (RFC5424) log message. If that fails, it parses the log message as a BSD-syslog (RFC3164) log message. Note that syslog-ng PE takes the parsing-related options of the configuration into account: `flags()`, `keep-hostname()`, `recv-time-zone()`.

If `keep-hostname()` is set to `no`, syslog-ng PE ignores the hostname set in the message, and uses the IP address of the syslog-ng PE host as the hostname (to use the hostname instead of the IP address, set the `use-dns()` or `use-fqdn()` options in the Python source).

```
msg_ietf = LogMessage.parse('<165>1 2003-10-11T22:14:15.003Z
mymachine.example.com evntslog - ID47 [exampleSDID@32473 iut="3"
eventSource="Application" eventID="1011"] An application event log entry',
self.parse_options)
msg_bsd = LogMessage.parse('<34>Oct 11 22:14:15 mymachine su: \'su root\' failed
for lonvick on /dev/pts/8', self.parse_options)
```

`set_pri()` method

You can set the priority of the message with the `set_pri()` method.

```
msg.set_pri(165)
```

`set_timestamp()` method

You can use the `set_timestamp()` method to set the date and time of the log message.

```
timestamp = datetime.fromisoformat("2018-09-11T14:49:02.100+02:00")
msg.set_timestamp(timestamp) # datetime object, includes timezone information
```

In Python 2, timezone information cannot be attached to the datetime instance without using an external library. The syslog-ng PE represents naive datetime objects in UTC.

In Python 3, naive and timezone-aware datetime objects are both supported.

python() and python-fetcher() source options

The `python()` and `python-fetcher()` drivers have the following options.

class()

Type:	string
Default:	N/A

Description: The name of the Python class that implements the source, for example:

```
python(
       class("MyPythonSource")
 );
```

If you want to store the Python code in an external Python file, the `class()` option must include the name of the Python file containing the class, without the path and the .py extension, for example:

```
python(
       class("MyPythonfilename.MyPythonSource")
 );
```

For details, see Python code in external files

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.

- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

The flags and the hostname-related options (for example, `use-dns`) set in the configuration file influence the behavior of the `LogMessage.parse()` method of the Python source. They have no effect if you set the message or the hostname directly, without using `LogMessage.parse()`.

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

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

hostname) of the host sending the message to syslog-ng PE. For details on using name resolution in syslog-ng PE, see 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.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with flow-control.

options()

Type:	string
Default:	N/A

Description: This option allows you to pass custom values from the configuration file to the Python code. Enclose both the option names and their values in double-quotes. The Python code will receive these values during initialization as the `options` dictionary. For example, you can use this to set the IP address of the server from the configuration file, so it is not hard-coded in the Python object.

```
python(
       class("MyPythonClass")
       options(
              "host"  "127.0.0.1"
              "port"  "1883"
              "otheroption"  "value")
 );
```

For example, you can refer to the value of the host field in the Python code as `options["host"]`. Note that the Python code receives the values as strings, so you might have to cast them to the type required, for example: `int(options["port"])`

persist-name()

Type:	string
Default:	None

Description: If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

NOTE: Starting with 7.0.193.0.26, syslog-ng PE assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your syslog-ng PE configuration, add a unique `persist-name()` to each source or destination, otherwise syslog-ng PE will not start. For example:

```
log {
      source { python(class(PyNetworkSource) options("port" "8080")
persist-name("<unique-string>"); };
      source { python(class(PyNetworkSource) options("port" "8081")); };
   };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:

```
from syslogng import LogSource
  class PyNetworSource(LogSource):
    @staticmethod
    def generate_persist_name(options):
        return options["port"]
    def run(self):
        pass
    def request_exit(self):
        pass
```

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.

CAUTION:

This option is available only when using Python 3.

use-syslogng-pid()

Type:	yes | 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 PE process.

python-fetcher: writing fetcher-style Python sources

The Python source allows you to write your own source in Python.

You can write two different type of sources in Python:

- Server-style sources that receives messages. Write server-style sources if you want to use an event-loop based, nonblocking server framework in Python, or if you want to implement a custom loop.

- Fetcher-style sources that actively fetch messages. In general, write fetcher-style sources (for example, when using simple blocking APIs), unless you explicitly need a server-style source.

This section describes fetcher-style sources. For details on server-style sources, see python: writing server-style Python sources.

The following points apply to using Python blocks in syslog-ng PE in general:

- Only the default Python modules are available (that is, you cannot import external Python modules, and One Identity does not support using external Python modules).
- The syslog-ng PE application uses its own Python interpreter (shipped with the default syslog-ng PE installation) instead of the system's Python interpreter.
- The syslog-ng PE application is shipped with Python version 3.8.
- The Python block must be a top-level block in the syslog-ng PE configuration file.
- If you store the Python code in a separate Python file and only include it in the syslog-ng PE configuration file, make sure that the PYTHON_PATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start syslog-ng PE manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`

  In production, when syslog-ng PE starts on boot, you must configure your startup script to include the Python path. The exact method depends on your operating system. For recent Red Hat Enterprise Linux, Fedora, and CentOS distributions that use systemd, the `systemctl` command sources the `/etc/sysconfig/syslog-ng` file before starting syslog-ng PE. (On openSUSE and SLES, `/etc/sysconfig/syslog` file.) Append the following line to the end of this file: `PYTHONPATH="<path-to-your-python-file>"`, for example, `PYTHONPATH="/opt/syslog-ng/etc"`
- The Python object is initiated every time when syslog-ng PE is started or reloaded.

  CAUTION:

  If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.
- The Python block can contain multiple Python functions.
- Using Python code in syslog-ng PE can significantly decrease the performance of syslog-ng PE, especially if the Python code is slow. In general, the features of syslog-ng PE are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The syslog-ng PE application does not do any of this.
- Python error messages are available in the `internal()` source of syslog-ng PE.
- You can access the name-value pairs of syslog-ng PE directly through a message object or a dictionary.

- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of syslog-ng PE. For details, see Logging from your Python code.

- **Support disclaimer**

  CAUTION:

  This is a Preview Feature, which provides an insight to planned enhancements to functionality in the product. Consider this Preview Feature a work in progress, as it may not represent the final design and functionality.

  This feature has completed QA release testing, but its full impact on production systems has not been determined yet, and potential future changes in functionality and the user interface may result in compatibility issues in your current settings.

  One Identity recommends the following:

  - **Consider the potential risks when using this functionality in a production environment.**
  - **Consider the Support Policy on Product Preview Features before using this functionality in a production environment.**
  - **Closely and regularly keep track of official One Identity announcements about potential changes in functionality and the user interface. If these potential changes affect your configuration, check the changes you have to make in your configuration, otherwise your syslog-ng PE application may not start after upgrade.**
  - **Always perform tests prior to upgrades in order to avoid the risks mentioned.**

  However, you are welcome to try this feature and if you have any feedback, Contact One Identity.

  Support Policy on Product Preview Features

  The One Identity Support Team will:

  - **Accept and review each service request opened regarding a Preview Feature.**
  - **Consider all service requests relating to a Preview Features as severity level 3.**
  - **Provide best effort support to resolve any issues relating to a Preview Feature.**
  - **Work with customers to log any product defects or enhancements relating to Preview Features.**
  - **Not accept requests for escalations regarding Preview Features.**
  - **Not provide after-hours support for Preview Features.**

Using Python in syslog-ng PE is recommended only if you are familiar with both Python and syslog-ng PE. One Identity is not responsible for the quality, resource requirements, or any bugs in the Python code, nor any syslog-ng PE crashes, message losses, or any other damage caused by the improper use of this feature, unless explicitly stated in a contract with One Identity.

Declaration

Python sources consist of two parts. The first is a syslog-ng PE source object that you define in your syslog-ng PE configuration and use in the log path. This object references a Python class, which is the second part of the Python source. The Python class receives or fetches the log messages, and can do virtually anything that you can code in Python. You can either embed the Python class into your syslog-ng PE configuration file, or store it in an external Python file.

```
source <name_of_the_python_source>{
    python-fetcher(
        class("<name_of_the_python_class_executed_by_the_source>")
    );
};

python {
from syslogng import LogFetcher
from syslogng import LogMessage

class <name_of_the_python_class_executed_by_the_source>(LogFetcher):
    def init(self, options): # optional
        print("init")
        print(options)
        return True

    def deinit(self): # optional
        print("deinit")

    def open(self): # optional
        print("open")
        return True

    def fetch(self): # mandatory
        print("fetch")
        # return LogFetcher.FETCH_ERROR,
        # return LogFetcher.FETCH_NOT_CONNECTED,
        return LogFetcher.FETCH_SUCCESS, msg

    def request_exit(self):
        print("request_exit")
        # If your fetching method is blocking, do something to break it
```

```
        # For example, if it reads a socket: socket.shutdown()

    def close(self): # optional
        print("close")
};
```

Methods of the python-fetcher() source

Fetcher-style Python sources must be inherited from the `syslogng.LogFetcher` class, and must implement at least the `fetch` method. Multiple inheritance is allowed, but only for pure Python super classes.

For fetcher-style Python sources, syslog-ng PE handles the event loop and the scheduling automatically. You can use simple blocking server/client libraries to receive or fetch logs.

You can retrieve messages using the `fetch()` method.

`init(self, options)` method (optional)

The syslog-ng PE application initializes Python objects every time when it is started or reloaded. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your source to work.

> **CAUTION:**
>
> **If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.**

When this method returns with False, syslog-ng PE does not start. It can be used to check options and return False when they prevent the successful start of the source.

`options`: This optional argument contains the contents of the `options()` parameter of the syslog-ng PE configuration object as a Python dictionary.

`open(self)` method (optional)

The `open(self)` method opens the resources required for the source, for example, it initiates a connection to the target service. It is called after `init()` when syslog-ng PE is started or reloaded. If `fetch()` returns with an error, syslog-ng PE calls the `close()` and `open()` methods before trying to fetch a new message.

If `open()` fails, it should return the False value. In this case, syslog-ng PE retries it every `time-reopen()` seconds. By default, this is 1 second for Python sources and destinations, the value of `time-reopen()` is not inherited from the global option. For details, see Error handling in the python() destination.

`fetch(self)` method (mandatory)

Use the `fetch` method to fetch messages and pass them to the log paths.

For details on parsing messages, see Python LogMessage API.

The `fetch` method must return one of the following values:

- `LogFetcher.FETCH_ERROR`: Fetching new messages failed, syslog-ng PE calls the `close` and `open` methods.
- `LogFetcher.FETCH_NOT_CONNECTED`: Could not access the source, syslog-ng PE calls the `open` method.
- `LogFetcher.FETCH_SUCCESS, msg`: Post the message returned as the second argument.

`request_exit(self)` method (optional)

If you use blocking operations within the `fetch()` method, use `request_exit()` to interrupt those operations (for example, to shut down a socket), otherwise syslog-ng PE is not able to stop. Note that syslog-ng PE calls the `request_exit` method from a thread different from the source thread.

`close(self)` method (optional)

Close the connection to the target service. Usually it is called right before `deinit()` when stopping or reloading syslog-ng PE. It is also called when `fecth()` fails.

The `deinit(self)` method (optional)

This method is executed when syslog-ng PE is stopped or reloaded. This method does not return a value.

> **CAUTION:**
>
> **If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.**

For the list of available optional parameters, see python() and python-fetcher() source options.

Python LogMessage API

The LogMessage API allows you to create LogMessage objects in Python sources, parse syslog messages, and set the various fields of the log message.

`LogMessage()` method: Create log message objects

You can use the `LogMessage()` method to create a structured log message instance. For example:

```
from syslogng import LogMessage

msg = LogMessage() # Initialize an empty message with default values (recvd
timestamp, rcptid, hostid, ...)
msg = LogMessage("string or bytes-like object") # Initialize a message and set
its ${MESSAGE} field to the specified argument
```

You can also explicitly set the different values of the log message. For example:

```
msg["MESSAGE"] = "message"
msg["HOST"] = "hostname"
```

You can set certain special field (timestamp, priority) by using specific methods.

Note the following points when creating a log message:

- When setting the hostname, syslog-ng PE takes the following hostname-related options of the configuration into account: `chain-hostnames()`, `keep-hostname()`, `use-dns()`, and `use-fqdn()`.
- Python sources ignore the `log-msg-size()` option.
- The syslog-ng PE application accepts only one message from every `LogSource::post_message()` or `fetch()` call, batching is currently not supported. If your Python code accepts batches of messages, you must pass them to syslog-ng PE one-by-one. Similarly, if you need to split messages in the source, you must do so in your Python code, and pass the messages separately.
- Do not reuse or store LogMessage objects after posting (calling `post_message()`) or returning the message from `fetch()`.

`parse()` method: Parse syslog messages

The `parse()` method allows you to parse incoming messages as syslog messages. By default, the `parse()` method attempts to parse the message as an IETF-syslog (RFC5424) log message. If that fails, it parses the log message as a BSD-syslog (RFC3164) log message. Note that syslog-ng PE takes the parsing-related options of the configuration into account: `flags()`, `keep-hostname()`, `recv-time-zone()`.

If `keep-hostname()` is set to `no`, syslog-ng PE ignores the hostname set in the message, and uses the IP address of the syslog-ng PE host as the hostname (to use the hostname instead of the IP address, set the `use-dns()` or `use-fqdn()` options in the Python source).

```
msg_ietf = LogMessage.parse('<165>1 2003-10-11T22:14:15.003Z
mymachine.example.com evntslog - ID47 [exampleSDID@32473 iut="3"
eventSource="Application" eventID="1011"] An application event log entry',
self.parse_options)
msg_bsd = LogMessage.parse('<34>Oct 11 22:14:15 mymachine su: \'su root\' failed
for lonvick on /dev/pts/8', self.parse_options)
```

`set_pri()` method

You can set the priority of the message with the `set_pri()` method.

```
msg.set_pri(165)
```

`set_timestamp()` method

You can use the `set_timestamp()` method to set the date and time of the log message.

```
timestamp = datetime.fromisoformat("2018-09-11T14:49:02.100+02:00")
msg.set_timestamp(timestamp) # datetime object, includes timezone information
```

In Python 2, timezone information cannot be attached to the datetime instance without using an external library. The syslog-ng PE represents naive datetime objects in UTC.

In Python 3, naive and timezone-aware datetime objects are both supported.

python() and python-fetcher() source options

The `python()` and `python-fetcher()` drivers have the following options.

class()

Type:	string
Default:	N/A

Description: The name of the Python class that implements the source, for example:

```
python(
       class("MyPythonSource")
 );
```

If you want to store the Python code in an external Python file, the `class()` option must include the name of the Python file containing the class, without the path and the .py extension, for example:

```
python(
       class("MyPythonfilename.MyPythonSource")
 );
```

For details, see Python code in external files

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.

- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

The flags and the hostname-related options (for example, `use-dns`) set in the configuration file influence the behavior of the `LogMessage.parse()` method of the Python source. They have no effect if you set the message or the hostname directly, without using `LogMessage.parse()`.

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

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

hostname) of the host sending the message to syslog-ng PE. For details on using name resolution in syslog-ng PE, see 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.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with flow-control.

options()

Type:	string
Default:	N/A

Description: This option allows you to pass custom values from the configuration file to the Python code. Enclose both the option names and their values in double-quotes. The Python code will receive these values during initialization as the `options` dictionary. For example, you can use this to set the IP address of the server from the configuration file, so it is not hard-coded in the Python object.

```
python(
       class("MyPythonClass")
       options(
              "host"  "127.0.0.1"
              "port"  "1883"
              "otheroption"  "value")
 );
```

For example, you can refer to the value of the host field in the Python code as `options["host"]`. Note that the Python code receives the values as strings, so you might have to cast them to the type required, for example: `int(options["port"])`

persist-name()

Type:	string
Default:	None

Description: If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

NOTE: Starting with 7.0.193.0.26, syslog-ng PE assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your syslog-ng PE configuration, add a unique `persist-name()` to each source or destination, otherwise syslog-ng PE will not start. For example:

```
log {
      source { python(class(PyNetworkSource) options("port" "8080")
persist-name("<unique-string>"); };
      source { python(class(PyNetworkSource) options("port" "8081")); };
   };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:

```
from syslogng import LogSource
  class PyNetworSource(LogSource):
    @staticmethod
    def generate_persist_name(options):
        return options["port"]
    def run(self):
        pass
    def request_exit(self):
        pass
```

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.

CAUTION:

This option is available only when using Python 3.

use-syslogng-pid()

Type:	yes | 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 PE process.

`snmptrap`: Read Net-SNMP traps

Using the `snmptrap()` source, you can read and parse the SNMP traps of the Net-SNMP's `snmptrapd` application. syslog-ng PE can read these traps from a log file, and extract their content into name-value pairs, making it easy to forward them as a structured log message (for example, in JSON format). The syslog-ng PE application automatically adds the `.snmp.` prefix to the name of the fields the extracted from the message.

The `snmptrap()` source is available in syslog-ng PE version 3.107.0.3 and later.

Limitations

- The `snmptrap()` source has only the options listed in snmptrap() source options. Other options commonly available in other source drivers are not supported.
- In addition to traps, the log of `snmptrapd` may contain other messages (for example, daemon start/stop information, debug logs) as well. Currently syslog-ng PE discards these messages.
- Because of a bug, `snmptrapd` does not escape String values in the VarBindList if it can resolve an OID to a symbolic name. As a result, syslog-ng PE cannot process traps that contain the = in the value of the string. To overcome this problem, disable resolving OIDs in `snmptrapd`.
- The colon (`:`) character is commonly used in SNMP traps. However, this character cannot be used in the name of syslog-ng PE macros (name-value pairs). Therefore, the syslog-ng PE application automatically replaces all consecutive `:` characters with a single underscore (`_`) character. For example, you can reference the value of the `NET-SNMP-EXAMPLES-MIB::netSnmpExampleString` key using the `${NET-SNMP-EXAMPLES-MIB_netSnmpExampleString}` macro.

  Note that this affects only name-value pairs (macros). The generated message always contains the original name of the key.

Prerequisites

- Configure `snmptrapd` to log into a file.
- If you use SMIv1 traps, include the following format string in the configuration file of `snmptrapd`:

  ```
  format1 %.4y-%.2m-%.2l %.2h:%.2j:%.2k %B [%b]: %N\n\t%W Trap (%q)
  Uptime: %#T\n%v\n
  ```

- If you use SMIv2 traps, use the default format. The `snmptrap()` source of syslog-ng PE expects this default format:

  ```
  format2 %.4y-%.2m-%.2l %.2h:%.2j:%.2k %B [%b]:\n%v\n
  ```

- Beacause of an `snmptrapd` bug, if you specify the filename in the configuration file with `logOption`, you must also specify another output as a command line argument (-Lf, -Ls). Otherwise, `snmptrapd` will not apply the the trap format.

To use the `snmptrap()` driver, the `scl.conf` file must be included in your syslog-ng PE configuration:

```
@include "scl.conf"
```

Example: Using the snmptrap() driver

A sample `snmptrapd` configuration:

```
authCommunity log,execute,net public
format1 %.4y-%.2m-%.2l %.2h:%.2j:%.2k %B [%b]: %N\n\t%W Trap (%q) Uptime:
%#T\n%v\n
outputOption s
```

Starting `snmptrapd`: `snmptrapd -A -Lf /var/log/snmptrapd.log`

Sending a sample V2 trap message: `snmptrap -v2c -c public 127.0.0.1 666 NET-SNMP-EXAMPLES-MIB::netSnmpExampleHeartbeatNotification netSnmpExampleHeartbeatRate i 60 netSnmpExampleString s "string"`. From this trap, syslog-ng PE receives the following input:

```
2017-05-23 15:29:40 localhost [UDP: [127.0.0.1]:59993->[127.0.0.1]:162]:
SNMPv2-SMI::mib-2.1.3.0 = Timeticks: (666) 0:00:06.66   SNMPv2-
SMI::snmpModules.1.1.4.1.0 = OID: NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatNotification     NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatRate = INTEGER: 60        NET-SNMP-EXAMPLES-
MIB::netSnmpExampleString = STRING: string
```

The following syslog-ng PE configuration sample uses the default settings of the driver, reading SNMP traps from the `/var/log/snmptrapd.log` file, and writes the log messages generated from the traps into a file.

```
@include  "scl.conf"
 log  {
       source  {
             snmptrap(filename("/var/log/snmptrapd.log"));
       };
       destination  {
             file("/var/log/example.log");
       };
 };
```

From the trap, syslog-ng PE writes the following into the log file:

```
May 23 15:29:40 myhostname snmptrapd: hostname='localhost', transport_
info='UDP: [127.0.0.1]:59993->[127.0.0.1]:162', SNMPv2-SMI::mib-2.1.3.0='
(666) 0:00:06.66', SNMPv2-SMI::snmpModules.1.1.4.1.0='NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatNotification', NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatRate='60', NET-SNMP-EXAMPLES-
MIB::netSnmpExampleString='string'
```

Using the same input trap, the following configuration example formats the SNMP traps as JSON messages.

```
@include "scl.conf"
log {
      source {
            snmptrap(
                  filename("/var/log/snmptrapd.log")
                  set-message-macro(no)
            );
      };

      destination {
            file("/var/log/example.log"
                  template("$(format-json --scope dot-nv-pairs)\n")
            );
      };
};
```

The previous trap formatted as JSON:

```
{
   "_snmp":{
      "transport_info":"UDP: [127.0.0.1]:59993->[127.0.0.1]:162",
      "hostname":"localhost",
      "SNMPv2-SMI_snmpModules":{
         "1":{
            "1":{
               "4":{
                  "1":{
                     "0":"NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatNotification"
                  }
               }
            }
         }
      },
      "SNMPv2-SMI_mib-2":{
         "1":{
            "3":{
               "0":"(666) 0:00:06.66"
            }
         }
```

```
    },
    "NET-SNMP-EXAMPLES-MIB_netSnmpExampleString":"string",
    "NET-SNMP-EXAMPLES-MIB_netSnmpExampleHeartbeatRate":"60"
  }
}
```

snmptrap() source options

The `snmptrap()` driver has the following options. Only the `filename()` option is required, the others are optional.

filename()

Type:	path
Default:	

Description: The log file of `snmptrapd`. The syslog-ng PE application reads the traps from this file.

In addition to traps, the log of `snmptrapd` may contain other messages (for example, daemon start/stop information, debug logs) as well. Currently syslog-ng PE discards these messages.

persist-name()

Type:	string
Default:	None

*Description:*If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

Default value: `.snmp.` option.

set-message-macro()

Type:	yes|no
Default:	yes

Description: The `snmptrap()` source automatically parses the traps into name-value pairs, so you can handle the content of the trap as a structured message. Consequently, you might not even need the `${MESSAGE}` part of the log message. If `set-message-macro()` is set to `no`, syslog-ng PE leaves the `${MESSAGE}` part empty. If `set-message-macro()` is set to `yes`, syslog-ng PE generates a regular log message from the trap.

`syslog`: 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 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 syslog() source options.

Declaration

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

Example: 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 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 this 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.

NOTE: To receive UDP messages at very high message rate, you can use the `udp-balancer()` source. For details, see udp-balancer: Receiving UDP messages at very high rate.

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 Message size and encoding.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog

messages). If theBOM[1]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 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
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 `ip-protocol(4)`.

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)`. You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

---

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

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source  s_network_tcp  {
       network(
             transport("tcp")
             ip("::")
             ip-protocol(6)
             port(601)
       );
 };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to sources should be closed when syslog-ng is 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 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
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:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with 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: 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 `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections.

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. 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
Default:	In case of TCP transport: 514 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
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 this 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
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:	altp, udp, tcp, tls, proxied_tcp, or proxied_tls
Default:	tcp

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

For detailed information about how from version 7.0.23 onwards, syslog-ng PE supports the `proxied_tcp` parameter and the `proxied_tls` parameter, see Proxy Protocol support.

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 this 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.

trim-large-messages()

Type:	yes|no
Default:	Use the global `trim-large-messages()` option, which defaults to `no`.

Description: Determines what syslog-ng PE does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, syslog-ng PE drops the incoming log message.
- If set to `yes`, syslog-ng PE trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter  f_trimmed  {
        tags("trimmed");
  };
  ```

  If syslog-ng PE trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in syslog-ng PE version 7.0.143.21 and later.

Uses the value of the global option if not specified.

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 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: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

TIP: Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.

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 | 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 PE process.

`system`: Collecting the system-specific log messages of a platform

Starting with version 4 F13.2, 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 4.1-5.03.2-3.3 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: The default configuration file of syslog-ng PE. Starting with syslog-ng PE version 3.6, you can use the `system-expand` command-line utility (which is a shell script, located in the `modules/system-source/` directory) to display the configuration that the `system()` source will use.

CAUTION:

If syslog-ng PE does not recognize the platform it is installed on, it does not add any sources.

Starting with version 3.67.0, syslog-ng PE parses messages complying with the Splunk Common Information Model (CIM) and marked with `@cim` as JSON messages (for example, the ulogd from the netfilter project can emit such messages). That way, you can forward such messages without losing any information to CIM-aware applications (for example, Splunk).

Table 9: Sources automatically added by syslog-ng Premium Edition

Platform	Message source
AIX	`unix-dgram("/dev/log");`

FreeBSD	`unix-dgram("/var/run/log");`
	`unix-dgram("/var/run/logpriv" perm(0600));`
	`file("/dev/klog" follow-freq(0) program-override("kernel") flags(no-parse));`
	For FreeBSD versions earlier than 9.1, `follow-freq(1)` is used.

GNU/kFreeBSD	`unix-dgram("/var/run/log");` `file("/dev/klog" follow-freq(0) program-override("kernel"));`
HP-UX	`pipe("/dev/log" pad-size(2048));`

Linux	`unix-dgram("/dev/log");`
	`file("/proc/kmsg" program-override("kernel") flags(kernel));`
	Note that on Linux, the `so-rcvbuf()` option of the `system()` source is automatically set to 8192.
	If the host is running under systemd, syslog-ng PE reads directly from the systemd journal file using the `systemd-journal()` source.
	If the kernel of the host is version 3.5 or newer, and `/dev/kmsg` is seekable, syslog-ng PE will use that instead of `/proc/kmsg`, using the `multi-line-mode(indented)`, `keep-timestamp(no)`, and the `format(linux-kmsg)` options.
	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.

`systemd-journal`: 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. By default, syslog-ng PE adds the `.journald.` prefix to the name of every parsed value.

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 is more than one `systemd-journal()` source configured, syslog-ng PE will not start.

Declaration

```
systemd-journal(options);
```

Example: 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: 7.0

source s_journald {
      systemd-journal(prefix(".SDATA.journald."));
};

destination d_network {
      syslog("server.host");
};

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

Example: Filtering for a specific field using the systemd-journal () driver

```
@version: 7.0

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: Sending all fields in value-pairs using the systemd-journal() driver

```
@version: 7.0

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:

Table 10: Predefined macros

Journald field	syslog-ng predefined macro
MESSAGE	$MESSAGE
_HOSTNAME	$HOST
_PID	$PID
_COMM or SYSLOG_IDENTIFIER	$PROGRAM If both _COMM and SYSLOG_IDENTIFIER exists, syslog-ng PE uses SYSLOG_IDENTIFIER
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
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:	.journald.

Description: If this option is set, every non-built-in mapped names get a prefix (for example: `".SDATA.journald."`). By default, syslog-ng PE adds the `.journald.` prefix to every value.

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 journal, if the journal 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.

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: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

TIP: Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.

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 | 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 PE process.

`systemd-syslog`: 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: Using the systemd-syslog() driver

```
@version: 7.0

source s_systemdd {
      systemd-syslog();
};

destination d_network {
      syslog("server.host");
};

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

tcp, tcp6, udp, udp6: 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 network: Collecting messages using the RFC3164 protocol (network() driver) and network: 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 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 network: Collecting messages using the RFC3164 protocol (network() driver) and network: 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 Change an old source driver to the network() driver.

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 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')
        )
    );
};
```

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 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')
             )
       );
 };
```

udp-balancer: Receiving UDP messages at very high rate

The `udp-balancer()` source allows you to use multiple CPU cores to process the incoming UDP messages at a very high message rate, depending on the available hardware resources, incoming message size, and your syslog-ng PE configuration. Note that this feature requires a Linux kernel that supports the SO_REUSEPORT kernel option, so it is supported only selected platforms.

The `udp-balancer()` source uses multiple CPU cores that listen on a single UDP port. This allows the source to scale to very high message rates. Using a single port is convenient, because you do not need to use load balancing on the client side. You can set the number of listeners using the `listeners()` option. For the best performance, the value of the `listeners()` option should be equal to the number of cores available on the host running syslog-ng PE.

When to use the udp-balancer() source

One Identity recommends using the `udp-balancer()` source if all of the following apply:

- You are running syslog-ng PE in relay or server mode on a platform that supports the `udp-balancer()` source. For details, see Limitations.
- You must receive UDP message at a high message rate, and for some reason you cannot use TCP connections instead.
- It is not critical for you to store the incoming messages in the order they were sent.

Limitations

- The `udp-balancer()` source is using eBPF, therefore it works only with kernels that support eBPF. From the platforms supported by syslog-ng PE, the following kernel

versions have been tested:

- Debian 10 (Debian Buster)
- Oracle Linux 7.7, kernel version 4.14.35
- Red Hat Enterprise Linux 7 is NOT supported, because its kernel version is too old.
- Red Hat Enterprise Linux 8, kernel version 4.18.
- SUSE Linux Enterprise Server 12.4, kernel version 4.12.14
- SUSE Linux Enterprise Server 15, kernel version 5.3.18
- Ubuntu 18.04 LTS (Bionic Beaver), kernel version 4.15
- Ubuntu 16.04 LTS (Xenial Xerus), kernel version 4.15

- Since the `udp-balancer()` source is multithreaded, there is no way to guarantee that the destination of the log path will receive the messages in the same order as the clients sent them. If you need to maintain the original order of the messages, make sure that the log messages contain a unique identifier (for example, a message ID or timestamp), and that your log processor (for example, a SIEM) can use this ID to reorder the messages.

Declaration

```
udp-balancer([options]);
```

By default, the `udp-balancer()` driver binds to `0.0.0.0`, meaning that it listens on every available IPv4 interface on the UDP/514 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: Using the udp-balancer() driver

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

```
source  s_network  {
       udp-balancer(
             ip("192.168.1.1")
       );
 };
```

Listening on the IPv6 localhost, port 2222:

```
source  s_network6  {
       udp-balancer(
              ip("::1")
              port(2222)
              ip-protocol(6)
       );
};
```

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

```
source  s_network  {
       udp-balancer(
              ip("127.0.0.1")
              flags(syslog-protocol)
       );
};
```

Using 8 listeners to receive logs on port 5514:

```
source  s_net  {
       udp-balancer(
              listeners(8)
              port(5514)
              log-fetch-limit(10000)
              log-iw-size(10000)
              );
};
```

For details on the options of the `udp-balancer()` source, see udp-balancer() source options.

udp-balancer() source options

The `udp-balancer()` 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 Message size and encoding.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.

- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

host-override()

Type:	string
Default:	

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

ip() or localip()

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

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
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 `ip-protocol(4)`.

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)`. You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source  s_network_tcp  {
       network(
             transport("tcp")
             ip("::")
             ip-protocol(6)
             port(601)
       );
 };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-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 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).

listeners()

Type:	number
Default:	<number-of-processors-in-the-host>

The `udp-balancer()` source uses multiple CPU cores that listen on a single UDP port. This allows the source to scale to very high message rates. Using a single port is convenient, because you do not need to use load balancing on the client side. You can set the number of listeners using the `listeners()` option. For the best performance, the value of the `listeners()` option should be equal to the number of cores available on the host running syslog-ng PE.

log-fetch-limit()

Type:	number
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:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with 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: 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 `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections.

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. 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
Default:	514

Description: The port number to bind to.

program-override()

Type:	string
Default:	

Description: Replaces the ${PROGRAM} part of the message with the parameter string. For example, to mark every message coming from the kernel, include the program-override ("kernel") option in the source containing /proc/kmsg.

so-rcvbuf()

Type:	number
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 this 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.

trim-large-messages()

Type:	yes|no
Default:	Use the global `trim-large-messages()` option, which defaults to `no`.

Description: Determines what syslog-ng PE does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, syslog-ng PE drops the incoming log message.
- If set to `yes`, syslog-ng PE trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter  f_trimmed  {
        tags("trimmed");
  };
  ```

  If syslog-ng PE trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in syslog-ng PE version 7.0.143.21 and later.

Uses the value of the global option if not specified.

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: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

TIP: Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.

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 | 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 PE process.

unix-stream, unix-dgram: 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 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: 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 files or 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 Message size and encoding.

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng PE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog

messages). If theBOM[1]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).

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

log-fetch-limit()

Type:	number
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:	100

Description: The size of the initial window, this value is used during flow control. For details on flow control, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

Type:	number (bytes)
Default:	Use the global `log-msg-size()` option, which defaults to `65536` (64 KiB).

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

Description: A string added to the beginning of every log message. It can be used to add an arbitrary string to any log source, though it is most commonly used for adding `kernel:` to the kernel messages on Linux. NOTE: This option is deprecated. Use `program-override()` instead.

max-connections()

Type:	number (simultaneous connections)
Default:	256

Description: Limits the number of simultaneously open connections. Cannot be used with `unix-dgram()`.

optional()

Type:	yes or no
Default:	

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

owner()

Type:	string
Default:	root

Description: Set the uid of the socket.

pad-size()

Type:	number
Default:	0

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

`windowsevent`: Collecting Windows event logs

Event log messages collected by the Windows Event Collector for syslog-ng PE use this special source. To collect Windows event log messages, include this source in one of your source statements.

The Windows Event Collector tool for syslog-ng PE collects the log messages of Windows-based hosts in Unix datagram sockets, and then forwards them to a syslog-ng PE server over HTTPS (using TLS encryption and mutual authentication). syslog-ng PE reads the log messages using the `windowsevent()` source, and then parses the logs into key-value paris using the XML parser.

The XML parser uses the list-handling functionality to handle lists in the XML. Note that you cannot disable the list-handling functionality for the `windowsevent()` source.

For more information, see Windows Event Collector Administration Guide.

Declaration

```
source  s_wec  {
       windowsevent(
          prefix(".windowsevent.")
          unix-domain-socket("`syslog-ng-root`/var/run/wec.sock")
       );
 };
```

Starting with version 7.0.13, the syslog-ng PE`windowsevent()` source can process XML arrays and make the elements of the arrays available as name-value pairs. For example, the following XML array becomes available as name-value pairs:

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <EventID>5059</EventID>
  </System>
  <EventData>
    <Data Name="SubjectUserSid">S-1-5-18</Data>
    <Data Name="SubjectUserName">WIN-K1678A68SQ6$</Data>
  </EventData>
```

From the previous example, the following name-value pairs become available: `${Event.System.EventID}` (5059), `${Event.EventData.SubjectUserSid}` (S-1-5-18), `${Event.EventData.SubjectUserName}` (WIN-K1678A68SQ6$).

NOTE: The name-value pairs are only created from EventData.Data xml paths, that is, only for <Data> tags that are the children of an <EventData> tag and have the Name attribute.

If the array-like structure is not a Data tag under EventData tag, or it misses the Name attribute, then the regular XML-parser logic is used.

`windowsevent()` source options

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

prefix()

Type:	string
Default:	".windowsevent."

Description: The prefix that you wish to append to the key-value pairs.

If you want to send Windows event logs to SDATA, then set `prefix(".SDATA.")`. This can be useful, for example, when you forward Windows event logs to a syslog-ng Store Box.

unix-domain-socket()

Type:	string
Default:	/opt/syslog-ng/var/run/wec.sock

Description: The path to the Unix domain socket to read messages from.

Sending and storing log messages — destinations and destination drivers

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

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

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

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

Example: 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 Routing messages: log paths, flags, and filters.**
- **Hazard of data loss! If your log files are on an NFS-mounted network file system, see Using syslog-ng PE with NFS or CIFS (or SMB) file system for log files.**

The following destination driver groups are available in syslog-ng PE:

`elasticsearch2`: Sending messages directly to Elasticsearch version 2.0 or higher (DEPRECATED)

CAUTION:

This destination is deprecated and will be removed from a future version of syslog-ng PE. We recommend using the elasticsearch-http: Sending messages to Elasticsearch HTTP Event Collector destination instead.

Starting with version 5.63.7 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: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Note the following limitations when using the syslog-ng PE `elasticsearch2` destination:

- This destination is only supported on the Linux platform.

  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 Java 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: Sending log data to Elasticsearch version 2.x and above

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 transport 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("transport")
        custom-id("${UNIQID}")
        flush-limit("10000")
    );
};

log {
    source(s_syslog);
    destination(d_elastic);
    flags(flow-control);
};
```

Example: Sending log data to Elasticsearch using the HTTP REST API

The following example send messages to Elasticsearch over HTTP using its REST API:

```
@include "scl.conf"

source s_network {
      network(port(5555));
};

destination d_elastic {
      elasticsearch2(
            client-mode("http")
            cluster("es-syslog-ng")
            index("x201")
            cluster-url("http://192.168.33.10:9200")
            type("slng_test_type")
            flush-limit("0")
      );
};

log {
      source(s_network);
      destination(d_elastic);
      flags(flow-control);
};
```

- To install the software required for the `elasticsearch2` destination, see Prerequisites.
- For details on how the `elasticsearch2` destination works, see How syslog-ng PE interacts with Elasticsearch.
- For the list of options, see Elasticsearch2 destination options (DEPRECATED).

The `elasticsearch2()` driver is actually a reusable configuration snippet configured to receive log messages using the Java language-binding of syslog-ng PE. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of the elasticsearch configuration snippet on GitHub. For details on extending syslog-ng PE in Java, see the Getting started with syslog-ng development guide.

NOTE: If you delete all Java destinations from your configuration and reload syslog-ng, the JVM is not used anymore, but it is still running. If you want to stop JVM, stop syslog-ng and then start syslog-ng again.

Prerequisites

The following describes how to send messages from syslog-ng PE to Elasticsearch.

To send messages from syslog-ng PE to Elasticsearch

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. NOTE: This step is only required if you use the `elasticsearch2` destination in node mode or transport mode.

   Download the Elasticsearch libraries (version 2.x or newer from the 2.x line) from https://www.elastic.co/downloads/elasticsearch.One Identity tests the destination using Elasticsearch version 2.4.
3. NOTE: This step is only required if you use the `elasticsearch2` destination in node mode or transport mode.

   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 (about 1000 messages/second).
    - 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`, about 8000 messages/second), but the transfer is not reliable. In transport mode, over 5000-30000 messages can be lost before syslog-ng PE recognizes the error. In node mode, about 1000 messages can be lost.
    - 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 concurrent-requests().
- Version 3.107.0.3 and newer of syslog-ng PE automatically converts the timestamp (date) of the message to UTC, as needed by Elasticsearch and Kibana.

Client modes

The syslog-ng PE application can interact with Elasticsearch in the following modes of operation: http, https, node, searchguard, and transport.

- **HTTP mode**

  The syslog-ng PE application sends messages over HTTP using the REST API of Elasticsearch, and uses the `cluster_url()` and `cluster()` options from the syslog-ng PE configuration file. In HTTP mode, syslog-ng PE`elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTP mode is available in syslog-ng PE version 3.87 and newer.

  In version 3.107.0.3 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster_url()` and `server()` options. The syslog-ng PE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng PE does not have any direct influence on it.

- **HTTPS mode**

  The syslog-ng PE application sends messages over an encrypted and optionally authenticated HTTPS channel using the REST API of Elasticsearch, and uses the `cluster_url()` and `cluster()` options from the syslog-ng PE configuration file. In HTTPS mode, syslog-ng PE`elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTPS mode is available in syslog-ng PE version 3.107.0.3 and newer.

  This mode supports password-based and certificate-based authentication of the client, and can verify the certificate of the server as well.

  In version 3.107.0.3 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster_url()` and `server()` options. The syslog-ng PE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng PE does not have any direct influence on it.

- **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` parameter in the `.yml` file. For example: `path.home: /usr/share/elasticsearch`.

- **Search Guard mode**

  Use the Search Guard Elasticsearch plugin to encrypt and authenticate your connections to from syslog-ng PE to Elasticsearch 2.x. For Elasticsearch versions 5.x and newer, use HTTPS mode. For details on configuring Search Guard mode, see .

Elasticsearch2 destination options (DEPRECATED)

CAUTION:

This destination is deprecated and will be removed from a future version of syslog-ng PE. We recommend using the elasticsearch-http: Sending messages to Elasticsearch HTTP Event Collector destination instead.

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:	The syslog-ng PE module directory: /opt/syslog-ng/lib/syslog-ng/java-modules/

Description: The list of the paths where the required Java classes are located. For example, `class-path("/opt/syslog-ng/lib/syslog-ng/java-modules/:/opt/my-java-libraries/libs/")`. If you set this option multiple times in your syslog-ng PE configuration (for example, because you have multiple Java-based destinations), syslog-ng PE will merge every available paths to a single list.

Description: Include the path to the directory where you copied the required libraries (see Prerequisites), for example, `client_lib_dir(/user/share/elasticsearch-2.2.0/lib)`.

client-mode()

Type:	http | https | transport | node | searchguard
Default:	node

Description: Specifies the client mode used to connect to the Elasticsearch server, for example, `client-mode("node")`.

- **HTTP mode**

  The syslog-ng PE application sends messages over HTTP using the REST API of Elasticsearch, and uses the `cluster_url()` and `cluster()` options from the syslog-ng PE configuration file. In HTTP mode, syslog-ng PE`elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTP mode is available in syslog-ng PE version 3.87 and newer.

  In version 3.107.0.3 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster_url()` and `server()` options. The syslog-ng PE application will

use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng PE does not have any direct influence on it.

- **HTTPS mode**

  The syslog-ng PE application sends messages over an encrypted and optionally authenticated HTTPS channel using the REST API of Elasticsearch, and uses the `cluster_url()` and `cluster()` options from the syslog-ng PE configuration file. In HTTPS mode, syslog-ng PE`elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTPS mode is available in syslog-ng PE version 3.107.0.3 and newer.

  This mode supports password-based and certificate-based authentication of the client, and can verify the certificate of the server as well.

  In version 3.107.0.3 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster_url()` and `server()` options. The syslog-ng PE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng PE does not have any direct influence on it.

- **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` parameter in the `.yml` file. For example: `path.home: /usr/share/elasticsearch`.

- **Search Guard mode**

  Use the Search Guard Elasticsearch plugin to encrypt and authenticate your connections to from syslog-ng PE to Elasticsearch 2.x. For Elasticsearch versions 5.x and newer, use HTTPS mode. For details on configuring Search Guard mode, see .

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 resource().

cluster-url()

Type:	string
Default:	N/A

Description: Specifies the URL or the Elasticsearch cluster, for example, `cluster-url ("http://192.168.10.10:9200")"`). Note that this option works only in HTTP mode: `client_mode(http)`

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 flush-limit().

CAUTION:

Hazard of data loss! Using the `concurrent-requests()` option increases the number of messages lost in case the Elasticsearch server becomes inaccessible.

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 use-uniqid().)

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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 (about 1000 messages/second).

- 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`, about 8000 messages/second), but the transfer is not reliable. In transport mode, over 5000-30000 messages can be lost before syslog-ng PE recognizes the error. In node mode, about 1000 messages can be lost.
- 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 concurrent-requests().

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

http-auth-type()

Type:	none | basic | clientcert
Default:	none

Description: Determines how syslog-ng PE authenticates to the Elasticsearch server. Depending on the value of this option, you might have to set other options as well. Possible values:

- `none`: Connect to the Elasticsearch server without authentication.
- `basic`: Use password authentication. Also set the `http-auth-type-basic-username` and `http-auth-type-basic-password` options.
- `clientcert`: Use a certificate to authenticate. The certificate must be available in a Java keystore. Also set the `java-keystore-filepath` and `java-keystore-password` options.

This option is used only in HTTPS mode: `client_mode("https")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Example: HTTPS authentication examples

The following simple examples show the different authentication modes.

Simple password authentication:

```
destination d_elastic {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("basic")
              http-auth-type-basic-username("example-username")
              http-auth-type-basic-password("example-password")
       );
};
```

Certificate authentication:

```
destination d_elastic {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("clientcert")
              java-keystore-filepath("<path-to-your-java-keystore>.jks")
              java-keystore-password("password-to-your-keystore")
       );
};
```

Verify the certificate of the Elasticsearch server without authentication:

```
destination  d_elastic  {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("none")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the syslog-ng PE client and the Elasticsearch server):

```
destination  d_elastic  {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("clientcert")
              java-keystore-filepath("<path-to-your-java-keystore>.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

http-auth-type-basic-password()

Type:	string
Default:	N/A

Description: The password to use for password-authentication on the Elasticsearch server. You must also set the `http-auth-type-basic-username` option.

This option is used only in HTTPS mode with basic authentication: `client_mode ("https")` and `http-auth-type("basic")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Simple password authentication:

```
destination d_elastic {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("basic")
              http-auth-type-basic-username("example-username")
              http-auth-type-basic-password("example-password")
       );
};
```

http-auth-type-basic-username()

Type:	string
Default:	N/A

Description: The username to use for password-authentication on the Elasticsearch server. You must also set the `http-auth-type-basic-password` option.

This option is used only in HTTPS mode with basic authentication: `client_mode ("https")` and `http-auth-type("basic")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Simple password authentication:

```
destination d_elastic {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("basic")
              http-auth-type-basic-username("example-username")
              http-auth-type-basic-password("example-password")
       );
};
```

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}")`.

java-keystore-filepath()

Type:	string
Default:	N/A

Description: Path to the Java keystore file that stores the certificate that syslog-ng PE uses to authenticate on the Elasticsearch server. You must also set the `java-keystore-password` option.

To import a certificate into a Java keystore, use the appropriate tool of your Java implementation. For example, on Oracle Java, you can use the `keytool` utility:

```
keytool -import -alias ca -file <certificate-to-import> -keystore <keystore-to-import> -storepass <password-to-the-keystore>
```

This option is used only in HTTPS mode with basic authentication: `client_mode ("https")` and `http-auth-type("clientcert")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Certificate authentication:

```
destination d_elastic {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("clientcert")
              java-keystore-filepath("<path-to-your-java-keystore>.jks")
              java-keystore-password("password-to-your-keystore")
       );
};
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the syslog-ng PE client and the Elasticsearch server):

```
destination  d_elastic  {
       elasticsearch2(
             client-mode("https")
             cluster("es-syslog-ng")
             index("x201")
             cluster-url("http://192.168.33.10:9200")
             type("slng_test_type")
             flush-limit("0")
             http-auth-type("clientcert")
             java-keystore-filepath("<path-to-your-java-keystore>.jks")
             java-keystore-password("password-to-your-keystore")
             java-truststore-filepath("<path-to-your-java-keystore>.jks")
             java-truststore-password("password-to-your-keystore")
       );
 };
```

java-keystore-password()

Type:	string
Default:	N/A

Description: The password of the Java keystore file set in the `java-keystore-filepath` option.

To import a certificate into a Java keystore, use the appropriate tool of your Java implementation. For example, on Oracle Java, you can use the `keytool` utility:

```
keytool -import -alias ca -file <certificate-to-import> -keystore <keystore-to-import> -storepass <password-to-the-keystore>
```

This option is used only in HTTPS mode with basic authentication: `client_mode("https")` and `http-auth-type("clientcert")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Certificate authentication:

```
destination d_elastic {
    elasticsearch2(
        client-mode("https")
        cluster("es-syslog-ng")
        index("x201")
        cluster-url("http://192.168.33.10:9200")
        type("slng_test_type")
        flush-limit("0")
```

```
        http-auth-type("clientcert")
        java-keystore-filepath("<path-to-your-java-keystore>.jks")
        java-keystore-password("password-to-your-keystore")
    );
};
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the syslog-ng PE client and the Elasticsearch server):

```
destination d_elastic {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("clientcert")
              java-keystore-filepath("<path-to-your-java-keystore>.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

java-truststore-filepath()

Type:	string
Default:	N/A

Description: Path to the Java keystore file that stores the CA certificate that syslog-ng PE uses to verify the certificate of the Elasticsearch server. You must also set the `java-truststore-password` option.

If you do not set the `java-truststore-filepath` option, syslog-ng PE does accepts any certificate that the Elasticsearch server shows. In this case, the identity of the server is not verified, only the connection is encrypted.

To import a certificate into a Java keystore, use the appropriate tool of your Java implementation. For example, on Oracle Java, you can use the `keytool` utility:

```
keytool -import -alias ca -file <certificate-to-import> -keystore <keystore-to-import> -storepass <password-to-the-keystore>
```

This option is used only in HTTPS mode: `client_mode("https")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Verify the certificate of the Elasticsearch server without authentication:

```
destination  d_elastic  {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("none")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the syslog-ng PE client and the Elasticsearch server):

```
destination  d_elastic  {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("clientcert")
              java-keystore-filepath("<path-to-your-java-keystore>.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

java-truststore-password()

Type:	string
Default:	N/A

Description: The password of the Java truststore file set in the `java-truststore-filepath` option.

To import a certificate into a Java keystore, use the appropriate tool of your Java implementation. For example, on Oracle Java, you can use the `keytool` utility:

```
keytool -import -alias ca -file <certificate-to-import> -keystore <keystore-to-import> -storepass <password-to-the-keystore>
```

This option is used only in HTTPS mode: `client_mode("https")`, and is available in syslog-ng PE version 3.107.0.3 and newer.

Verify the certificate of the Elasticsearch server without authentication:

```
destination  d_elastic  {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("none")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the syslog-ng PE client and the Elasticsearch server):

```
destination  d_elastic  {
       elasticsearch2(
              client-mode("https")
              cluster("es-syslog-ng")
              index("x201")
              cluster-url("http://192.168.33.10:9200")
              type("slng_test_type")
              flush-limit("0")
              http-auth-type("clientcert")
              java-keystore-filepath("<path-to-your-java-keystore>.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("<path-to-your-java-keystore>.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng PE configuration file.

For example:

```
jvm-options("-Xss1M -XX:+TraceClassLoading")
```

You can set this option only as a global option, by adding it to the `options` statement of the syslog-ng configuration file.

log-fifo-size()

Type:	number
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")`

In version 3.107.0.3 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster_url()` and `server()` options. The syslog-ng PE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng PE does not have any direct influence on it.

For example:

```
destination  d_elasticsearch  {
       elasticsearch2(
             client-lib-dir("/usr/share/elasticsearch/lib/")
             index("syslog-${YEAR}.${MONTH}.${DAY}")
             type("syslog")
             time-zone("UTC")
             client_mode("http")
             server("node01  node02")
             port(9200)
       );
 };
```

skip-cluster-health-check()

Type:	yes|no
Default:	no

Description: By default, when connecting to an Elasticsearch cluster, syslog-ng PE checks the state of the cluster. If the primary shards of the cluster are not active, syslog-ng PE will not send messages, but wait for them to become active. To disable this health check and send the messages to Elasticsearch anyway, use the `skip-cluster-health-check(yes)` option in your configuration.

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 format-json.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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.

Version 3.107.0.3 and newer of syslog-ng PE automatically converts the timestamp (date) of the message to UTC, as needed by Elasticsearch and Kibana.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

type()

Type:	string
Default:	N/A

Description: The type of the index. For example, `type("test")`.

`elasticsearch-http`: Sending messages to Elasticsearch HTTP Event Collector

Version 7.0.143.21 of syslog-ng PE can directly post log messages to an Elasticsearch deployment using the Elasticsearch Bulk API over the HTTP and Secure HTTP (HTTPS) protocols.

HTTPS connection, as well as password- and certificate-based authentication is supported. The content of the events is sent in JSON format.

NOTE: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Declaration

```
d_elasticsearch_http  {
       elasticsearch-http(
             index("<elasticsearch-index-to-store-messages>")
             url("https://your-elasticsearch-server1:9200/_bulk"  "https://your-
elasticsearch-server2:9200/_bulk")
             type("<type-of-the-index>")
       );
 };
```

Example: Sending log data to Elasticsearch

The following example defines a `elasticsearch-http()` destination, with only the required options.

```
destination  d_elasticsearch_http  {
       elasticsearch-http(
             index("<name-of-the-index>")
             type("<type-of-the-index>")
             url("http://my-elastic-server:9200/_bulk")
       );
 };


 log  {
       source(s_file);
       destination(d_elasticsearch_http);
       flags(flow-control);
 };
```

The following example uses mutually-authenticated HTTPS connection, templated index, and also sets the `type()` and some other options.

```
destination  d_elasticsearch_https  {
       elasticsearch-http(
             url("https://node01.example.com:9200/_bulk")
             index("test-${YEAR}${MONTH}${DAY}")
             time-zone("UTC")
             type("test")
             workers(4)
             batch-lines(16)
             timeout(10)
             tls(
                   ca-file("ca.pem")
                   cert-file("syslog_ng.crt.pem")
                   key-file("syslog_ng.key.pem")
                   peer-verify(yes)
             )
       );
 };
```

This driver is actually a reusable configuration snippet configured to send log messages using the `tcp()` driver using a template. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of this configuration snippet on GitHub.

Batch mode and load balancing

The `elasticsearch-http()` destination automatically sends multiple log messages in a single HTTP request, increasing the rate of messages that your Elasticsearch deployment can consume. For details on adjusting and fine-tuning the batch mode of the `elasticsearch-http()` destination, see the following section.

Batch size

The `batch-lines()`, `batch-lines()`, and `batch-timeout()` options of the destination determine how many log messages syslog-ng PE sends in a batch. The `batch-lines()` option determines the maximum number of messages syslog-ng PE puts in a batch in. This can be limited based on size and time:

- syslog-ng PE sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. This ensures that the destination receives every message in a timely manner even if suddenly there are no more messages.
- syslog-ng PE sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Example: HTTP batch mode

In the following example, a batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
destination  d_elasticsearch-http  {
       elasticsearch-http(url("http://your-elasticsearch-server:9200/_
bulk")
              index("<elasticsearch-index-to-store-messages>")
              type("")
              url("http://your-elasticsearch-server:9200/_bulk")
              batch-lines(100)
              batch-bytes(512Kb)
              batch-timeout(10000)
       );
 };
```

Load balancing between multiple Elasticsearch indexers

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

Example: HTTP load balancing

The following destination sends log messages to 3 different Elasticsearch indexer nodes. Each node is assigned a separate worker thread. A batch consists of 100

messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
destination  d_elasticsearch-http  {
       elasticsearch-http(url("http://your-elasticsearch-server1:9200/_
bulk"  "http://your-elasticsearch-server2:9200/_bulk"  "http://your-
elasticsearch-server3:9200/_bulk")
              batch-lines(100)
              batch-bytes(512Kb)
              batch-timeout(20000)
              persist-name("d_elasticsearch-http-load-balance")
       );
 };
```

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

elasticsearch-http destination options

The `elasticsearch-http` destination of syslog-ng PE can directly post log messages to an Elasticsearch deployment using the Elasticsearch Bulk API over the HTTP and Secure HTTP (HTTPS) protocols. The `elasticsearch-http` destination has the following options. The required options are: `index()`, `type()`, and `url()`.

This destination is available in syslog-ng PE version 7.0.143.21 and later.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, syslog-ng PE sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in syslog-ng PE version 3.197.0.12 and later.

For details on how this option influences batch mode, see Batch mode and load balancing on page 342

batch-lines()

Type:	number
Default:	25

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

For details on how this option influences batch mode, see Batch mode and load balancing on page 342

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

For details on how this option influences batch mode, see Batch mode and load balancing on page 342

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. For an example, see Configuring TLS on the syslog-ng clients. The syslog-ng PE application uses the CA certificates in this directory to validate the certificate of the peer.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
        elasticsearch-http(
                url("http://your-elasticsearch-server:9200/_bulk")
                type("")
                index("example-index")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the HTTPS server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
        elasticsearch-http(
                url("http://your-elasticsearch-server:9200/_bulk")
                type("")
                index("example-index")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 set in the `key-file()` option. The syslog-ng PE application uses this certificate to authenticate the syslog-ng PE client on the destination server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
       elasticsearch-http(
               url("http://your-elasticsearch-server:9200/_bulk")
               type("")
               index("example-index")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
                       peer-verify(yes|no)
                       ssl-version(<the permitted SSL/TLS version>)
               )
       );
};
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that syslog-ng PE uses

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")
```

For a list of available algorithms, execute the `openssl ciphers -v` command. The first column of the output contains the name of the algorithms to use in the `cipher-suite()` option, the second column specifies which encryption protocol uses the algorithm (for example, `TLSv1.2`). That way, the `cipher-suite()` also determines the encryption protocol used in the connection: to disable SSLv3, use an algorithm that is available only in TLSv1.2, and that both the client and the server supports. You can also specify the encryption protocols using ssl-options().

You can also use the following command to automatically list only ciphers permitted in a specific encryption protocol, for example, `TLSv1.2`:

```
echo "cipher-suite(\"$(openssl ciphers -v | grep TLSv1.2 | awk '{print $1}' |
xargs echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
        elasticsearch-http(
                url("http://your-elasticsearch-server:9200/_bulk")
                type("")
                index("example-index")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

custom-id()

Accepted values:	string
Default:	empty string

Description: Sets the specified value as the ID of the Elasticsearch index (`_id`).

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, syslog-ng PE separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option.

For details on how this option influences batch mode, see Batch mode and load balancing on page 342

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
              )
       );
};
```

hook-commands()

Description: This option makes it possible to execute external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to execute external applications.

Using the hook-commands() when syslog-ng PE starts or stops

To execute an external program when syslog-ng PE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed when syslog-ng PE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed when syslog-ng PE stops.

Using the hook-commands() when syslog-ng PE reloads

To execute an external program when the syslog-ng PE configuration is initiated or torn down (for example, on startup/shutdown or during a syslog-ng PE reload), use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.

Example: Using the `hook-commands()` with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically when syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow (otherwise the port is closed).

```
source  {
    network(transport(udp)
       hook-commands(
          startup("iptables  -I  LOGCHAIN  1  -p  udp  --dport  514  -j
ACCEPT")
          shutdown("iptables  -D  LOGCHAIN  1")
       ;)
    );
 };
```

index()

Accepted values:	string or template
Default:	None

Description: The name of the Elasticsearch index where Elasticsearch will store the messages received from syslog-ng PE. This option is mandatory for this destination.

You can use macros and template functions, but you must ensure that the resolved template contains only characters that Elasticsearch permits in the name of the index. The syslog-ng PE application does not validate the name of the index. For details on the characters permitted in the name of Elasticsearch indices, see the documentation of Elasticsearch.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

key-file()

Accepted values:	Filename
Default:	none

Description: Path and name of a file that contains a private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng PE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng PE client on the destination server.

This destination supports only unencrypted key files (that is, the private key cannot be password-protected).

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
        elasticsearch-http(
                url("http://your-elasticsearch-server:9200/_bulk")
                type("")
                index("example-index")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

password()

Type:	string
Default:	

Description: The password that syslog-ng PE uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	yes | no
Default:	yes

Description: Verification method of the peer. 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	*no (optional-untrusted)*	TLS-encryption	TLS-encryption	TLS-encryption
	yes (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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
       elasticsearch-http(
               url("http://your-elasticsearch-server:9200/_bulk")
               type("")
               index("example-index")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
```

```
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

persist-name()

Type:	string
Default:	None

Description: If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

The `proxy()` option enables you to configure the `elasticsearch-http` driver to use a specific HTTP proxy for all HTTP-based destinations, instead of using the proxy that is configured for the system.

If you do not set the `proxy()` option, the `elasticsearch-http` driver uses the `http_proxy` and `https_proxy` environment variables, as shown in CURLOPT_PROXY explained.

> NOTE: Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.

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.

To handle HTTP error responses, if the HTTP server returns 5xx codes, syslog-ng PE will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, syslog-ng PE will drop the messages.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_elasticsearch-http {
        elasticsearch-http(
                url("http://your-elasticsearch-server:9200/_bulk")
                type("")
                index("example-index")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

type()

Type:	string or template
Default:	N/A

Description: The type of the Elasticsearch index. Use an empty string to omit the type from the index, for example, `type("")`.

timeout()

Type:	number [seconds]
Default:	10

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded.

url()

Type:	URL or list of URLs, for example, url("site1" "site2")
Default:	N/A

Description: Specifies the hostname or IP address and optionally the port number of the Elasticsearch indexer. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://your-elasticsearch-indexer.server:8088/_bulk`

This option is mandatory for this destination.

Make sure that the URL ends with `_bulk`, this is the Elasticsearch API endpoint that properly parses the messages sent by syslog-ng PE.

In case the server on the specified URL returns a redirect request, syslog-ng PE automatically follows maximum 3 redirects. Only HTTP and HTTPS based redirections are supported.

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For

example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

user()

Type:	string
Default:	

Description: The username that syslog-ng PE uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	yes | no
Default:	no

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation.

workers()

Type:	integer
Default:	4

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

`file`: 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 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 file() destination options.

Declaration

```
file(filename options());
```

Example: Using the file() driver

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

Example: 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}]
${MESSAGE}\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 The syslog-ng manual page 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 when creating files or socket files.

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()`.

Starting with version 7.0.93.16, the default value of this option is -1, so syslog-ng PE does not change the ownership, unless explicitly configured to do so.

dir-perm()

Type:	number
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`).

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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, 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 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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
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.xsyslog-ng PE 3.3 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. Note that setting the global mark-mode() to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 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
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
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 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 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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

google_pubsub(): Sending logs to the Google Cloud Pub/Sub messaging service

From syslog-ng Premium Edition (syslog-ng PE) version 7.0.21, you can use the the `google_pubsub()` destination to generate your own messaging Google Pub/Sub infrastructure with syslog-ng PE as a "Publisher" entity, utilizing the HTTP REST interface of the service.

Similarly to syslog-ng PE's `stackdriver() destination`, the `google_pubsub()` destination is an asynchronous messaging service connected to Google's infrastructure.

For more information about Google Pub/Sub's messaging service, see What Is Pub/Sub?. The rest of this section and its subsections assume that you are familiar with the Google Pub/Sub messaging service, and its concepts and terminology.

Limitations

The current implementation of the `google_pubsub()` destination has the following limitations:

No message-based acknowledgement: While Google Pub/Sub acknowledges the batch of received messages, it also sends individual acknowledgement ID's to each message. However, we currently do not track individual messages inside Google Pub/Sub. Under normal operation circumstances, the lack of tracking individual messages has no effect on message delivery, and even allows flow control to work properly. However, in case of an error, the only solution is to repeat the entire batch, which can lead to message duplication in case Google Pub/Sub acknowledged part of the previous batch despite indicating an overall error.

NOTE: This behavior, called At-Least-Once delivery, means that if an error occurs, it is more acceptable to duplicate messages than to lose any of them.

NOTE: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

NOTE: The `google_pubsub()` destination can not fetch logs, only serve as a "Publisher" entity in the Google Pub/Sub service.

Configuring the google_pubsub() destination

From syslog-ng Premium Edition (syslog-ng PE) version 7.0.21, you can use the the `google_pubsub()` destination to generate your own messaging Google Pub/Sub infrastructure with syslog-ng PE as a "Publisher" entity, utilizing the HTTP REST interface of the service.

Similarly to syslog-ng PE's `stackdriver() destination`, the `google_pubsub()` destination is an asynchronous messaging service connected to Google's infrastructure.

For more information about Google Pub/Sub's messaging service, see What Is Pub/Sub?. The rest of this section and its subsections assume that you are familiar with the Google Pub/Sub messaging service, and its concepts and terminology.

Prerequisites

To configure the `google_pubsub()` destination, you must have each of the following:

- Appropriate credentials through a Google Cloud Pub/Sub service account.

  For more information, see Quickstart: building a functioning Pub/Sub system > Create service account credentials.

- A Google Pub/Sub topic and a Google Pub/Sub subscription.

  For more information, see Quickstart: building a functioning Pub/Sub system > Set up your Google Cloud project and Pub/Sub topic and subscriptions.

For more information, see Quickstart: building a functioning Pub/Sub system.

Declaration

```
destination d_google_pubsub {
    google_pubsub(
       credentials("<path-to-your-credentials-file>")
       gcp_auth_header_params(ca_file(<path-to-your-systems-root-certificate>))
       project(<the-name-of-your-pub-sub-project>)
       topic(<the-name-of-your-pub-sub-topic>)
    );
};
```

Configuration

The following example is a sample configuration for the `google_pubsub()` destination.

Example: configuring the `google_pubsub()` destination

In your syslog-ng PE configuration, you can use the `google_pubsub()` destination like this:

```
destination d_google_pubsub {
    google_pubsub(
       credentials("/home/user/service-account-key.json")
       gcp_auth_header_params(ca_file(<path-to-your-systems-root-
certificate>))
       project(mypubsubproject)
       topic(mytopic)
    );
};
```

NOTE: Similarly to the `stackdriver()` destination, while using the the `gcp_auth_header_params` parameter in the `google_pubsub()` destination, you must use underscore (_) in the options that syslog-ng PE passes directly to Google Pub/Sub).

NOTE: The `google_pubsub()` destination is a customized `http()` destination. As a result, the `google_pubsub()` destination accepts every `http()` option and its parameters (for example, the stackdriver destination options, with the exception of the Stackdriver-specific options). While using the options of the `http()` destination and their parameters with the `google_pubsub()`destination, the hyphen (`-`) and underscore (_) characters are

interchangeable.

Required options and parameters

The `google_pubsub()` destination has the following required options and parameters:

- `gcp_auth_header_params()`
- the `ca-dir()` parameter of the `gcp_auth_header_params()` option

  or

- the `ca-file()` parameter of the `gcp_auth_header_params()` option

NOTE: Specifying at least one of the `ca-dir()` or `ca-file()` parameters of the `gcp_auth_header_params()` option is mandatory. As a result, the `gcp_auth_header_params()` option is a required option.

google_pubsub() destination options

From syslog-ng Premium Edition (syslog-ng PE) version 7.0.21, you can use the the `google_pubsub()` destination to generate your own messaging Google Pub/Sub infrastructure with syslog-ng PE as a "Publisher" entity, utilizing the HTTP REST interface of the service.

The `google_pubsub()` destination has the following options.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, syslog-ng PE sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in syslog-ng PE version 7.0.12 and later.

batch-lines()

Type:	number [lines]
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

body()

Type:	string or template
Default:	

Description: The body of the HTTP request, for example, `body("${ISODATE} ${MESSAGE}")`. You can use strings, macros, and template functions in the body. If not set, it will contain the message received from the source by default.

body-prefix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts at the beginning of the body of the HTTP request, before the log message. Available in syslog-ng PE version 7.0.11 and later.

body-suffix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts to the end of the body of the HTTP request, after the log message. Available in syslog-ng PE version 7.0.11 and later.

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. For an example, see Configuring TLS on the syslog-ng clients. The syslog-ng PE application uses the CA certificates in this directory to validate the certificate of the peer.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
       http(
               url("http://127.0.0.1:8080")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
```

```
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the HTTPS server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 set in the `key-file()` option. The syslog-ng PE application uses this certificate to authenticate the syslog-ng PE client on the destination server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that syslog-ng PE uses

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")
```

For a list of available algorithms, execute the `openssl ciphers -v` command. The first column of the output contains the name of the algorithms to use in the `cipher-suite()` option, the second column specifies which encryption protocol uses the algorithm (for example, `TLSv1.2`). That way, the `cipher-suite()` also determines the encryption protocol used in the connection: to disable SSLv3, use an algorithm that is available only in TLSv1.2, and that both the client and the server supports. You can also specify the encryption protocols using ssl-options().

You can also use the following command to automatically list only ciphers permitted in a specific encryption protocol, for example, `TLSv1.2`:

```
echo "cipher-suite(\"$(openssl ciphers -v | grep TLSv1.2 | awk '{print $1}' |
xargs echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, syslog-ng PE separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option. Available in syslog-ng PE version 3.187.0.11 and later.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
              )
       );
 };
```

gcp_auth_header_params()

Description: Required option. The parameters of the `gcp_auth_header_params()` option are passed directly to the module that will authenticate syslog-ng PE to the Google server through the OAuth2.0 protocol. The `gcp_auth_header_params()` option has the following parameters:

ca-dir()	
Accepted values:	directory name

Default:	none

NOTE: At least one of the `ca-dir()` or `ca-file()` parameters of the `gcp_auth_header_params()` option is required.

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-file()

Accepted values:	Filename
Default:	none

NOTE: At least one of the `ca-dir()` or `ca-file()` parameters of the `gcp_auth_header_params()` option is required.

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the HTTPS server. 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.

scope()

Type:	OAuth 2.0 Scope
Default:	https://www.googleapis.com/auth/cloud-platform

Description: A mechanism in OAuth 2.0 to limit an application's access to a user's account.

For a detailed description of OAuth scopes, see OAuth Scopes.

timeout()

Type:	number [seconds]
Default:	0

Description: The timeout of the individual requests during API key renewal. The default value of `0` means there is no timeout set.

NOTE: The `timeout()` value is only used for authentication purposes, and it is independent from log message delivery.

headers()

Type:	string list
Default:	

Description: Custom HTTP headers to include in the request, for example, `headers ("HEADER1: header1", "HEADER2: header2")`. If not set, only the default headers are included, but no custom headers.

The following headers are included by default:

- X-Syslog-Host: <host>
- X-Syslog-Program: <program>
- X-Syslog-Facility: <facility>
- X-Syslog-Level: <loglevel/priority>

hook-commands()

Description: This option makes it possible to execute external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to execute external applications.

Using the hook-commands() when syslog-ng PE starts or stops

To execute an external program when syslog-ng PE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed when syslog-ng PE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed when syslog-ng PE stops.	

Using the hook-commands() when syslog-ng PE reloads

To execute an external program when the syslog-ng PE configuration is initiated or torn down (for example, on startup/shutdown or during a syslog-ng PE reload), use the

following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.	

Example: Using the `hook-commands()` with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically when syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow (otherwise the port is closed).

```
source  {
    network(transport(udp)
       hook-commands(
          startup("iptables  -I  LOGCHAIN  1  -p  udp  --dport  514  -j
ACCEPT")
          shutdown("iptables  -D  LOGCHAIN  1")
       ;)
    );
 };
```

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

key-file()

Accepted values:	Filename
Default:	none

Description: Path and name of a file that contains a private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng PE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng PE client on the destination server.

The `sentinel()` destination supports only unencrypted key files (that is, the private key cannot be password-protected).

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

method()

Type:	POST | PUT
Default:	POST

Description: Specifies the HTTP method to use when sending the message to the server.

password()

Type:	string
Default:	

Description: The password that syslog-ng PE uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	yes | no
Default:	yes

Description: Verification method of the peer. 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	*no (optional-untrusted)*	TLS-encryption	TLS-encryption	TLS-encryption
	yes (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.

persist-name()

Type:	string
Default:	None

*Description:*If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

The `proxy()` option enables you to configure the `google_pubsub()` driver to use a specific HTTP proxy for all HTTP-based destinations, instead of using the proxy that is configured for the system.

If you do not set the `proxy()` option, the `google_pubsub()` driver uses the `http_proxy` and `https_proxy` environment variables, as shown in CURLOPT_PROXY explained.

> NOTE: Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.

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.

To handle HTTP error responses, if the HTTP server returns 5xx codes, syslog-ng PE will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, syslog-ng PE will drop the messages.

service_endpoint()

Type:	Regional endpoint URL for Google Pub/Sub services. For example, `https://asia-south1-pubsub.googleapis.com`
Default:	`https://pubsub.googleapis.com/`

Description: Optional. When configured, the `google_pubsub()` destination can route requests to the regional service endpoint of your choice.

Syntax:

```
service_endpoint("https://asia-south1-pubsub.googleapis.com")
```

For the list of available endpoint URLs for the `google_pubsub()` destination, see the description and list of available Google Pub/Sub service endpoints.

For more information about including the `service_endpoint()` option in your configuration, see Available endpoints for the google_pubsub() destination.

NOTE: The `service_endpoint()` option is only available in syslog-ng PE version 7.0.24 and later.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

timeout()

Type:	number [seconds]
Default:	0

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded. By default, the timeout value is `0`, meaning that there is no timeout. Available in version 7.0.4 and later.
(missing or bad snippet)

url()

Type:	URL or list of URLs
Default:	http://localhost/

Description: Specifies the hostname or IP address, and optionally the port number of the web service that can receive log data through HTTP. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://127.0.0.1:8000`

In case the server on the specified URL returns a redirect request, syslog-ng PE automatically follows maximum 3 redirects. Only HTTP-based and HTTPS-based redirections are supported.

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

user-agent()

Type:	string
Default:	syslog-ng [version]/libcurl[version]

Description: The value of the USER-AGENT header in the messages sent to the server.

user()

Type:	string
Default:	

Description: The username that syslog-ng PE uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	yes | no
Default:	no

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

Available endpoints for the google_pubsub() destination

From syslog-ng Premium Edition (syslog-ng PE) version 7.0.21, you can use the the `google_pubsub()` destination to generate your own messaging Google Pub/Sub infrastructure with syslog-ng PE as a "Publisher" entity, utilizing the HTTP REST interface of the service.

Similarly to syslog-ng PE's `stackdriver() destination`, the `google_pubsub()` destination is an asynchronous messaging service connected to Google's infrastructure.

For more information about Google Pub/Sub's messaging service, see What Is Pub/Sub?. The rest of this section and its subsections assume that you are familiar with the Google Pub/Sub messaging service, and its concepts and terminology.

Available endpoints for the google_pubsub() destination

From syslog-ng PE version 7.0.24, the following endpoints are available for the `google_pubsub()` destination:

- **The global REST/HTTP endpoint (default)**

  By default, the `google_pubsub()` destination uses the global REST/HTTP endpoint, which automatically redirects requests to a nearby region.

  The default global REST/HTTP endpoint for Google Pub/Sub services, which is also the default configuration option for the `google_pubsub()`, destination, is `"https://pubsub.googleapis.com"`.

- **Your custom service endpoint (optional)**

  From version 7.0.24, the syslog-ng PE application supports including the `service_endpoint()` option to configure an additional service endpoint in addition to the default global REST/HTTP endpoint.

For more information, see the description and list of available Google Pub/Sub service endpoints.

> NOTE: Configuring the `service_endpoint()` option for the `google_pubsub()`, destination is optional, and it does not affect the default global REST/HTTP endpoint in any way.

Configuring the service-endpoint() option for the google_pubsub() destination

You can configure your `google_pubsub()` destination to route messages to regional service endpoints in addition to the default global REST/HTTP endpoint by including the `service_endpoint()` option in your configuration.

Example: configuring the service_endpoint() option for the google_pubsub() destination

You can include the `service_endpoint()` option for the `google_pubsub()` destination as follows:

```
destination  {
    google_pubsub(
        ...
        service_endpoint("https://southamerica-east1-
pubsub.googleapis.com")
    );
 };
```

CAUTION: When configuring the `service_endpoint()`option, make sure you include the `https://` part of the endpoint URL. To avoid mistakes, One Identity recommends that you copy the REST/HTTP endpoint column of your choice as-is, directly from the table of the listed available regional endpoints.

For example:

Figure 27: Available regional endpoint URLs for Google Pub/Sub

Region	REST/HTTP endpoint	gRPC Service
us-east1	https://us-east1-pubsub.googleapis.com	us-east1-pubsub.googleapis.com
us-east4	https://us-east4-pubsub.googleapis.com	us-east4-pubsub.googleapis.com
us-central1	https://us-central1-pubsub.googleapis.com	us-central1-pubsub.googleapis.com
us-west1	https://us-west1-pubsub.googleapis.com	us-west1-pubsub.googleapis.com
us-west2	https://us-west2-pubsub.googleapis.com	us-west2-pubsub.googleapis.com
us-west3	https://us-west3-pubsub.googleapis.com	us-west3-pubsub.googleapis.com
southamerica-east1	https://southamerica-east1-pubsub.googleapis.com	southamerica-east1-pubsub.googleapis.com
northamerica-northeast1	https://northamerica-northeast1-pubsub.googleapis.com	northamerica-northeast1-pubsub.googleapis.com

The example above illustrates the REST/HTTP endpoint URL of the `southamerica-east1` region, which is the same region that is included in the configuration example.

Error messages you may encounter while using the google_pubsub() destination

The following table describes the possible error messages that you may encounter while using the `google_pubsub()` destination.

status_code	Complete response while running with trace messages enabled	Possible reason(s)	Possible solution (s)
400	`"error": { "code": 400, "message": "The value for message_count is too large. You passed 1001 in the request, but the maximum value is 1000.",`	There are too many messages in one batch. Google Pub/Sub allows maximum 1000 messages per batch.	Decrease the value of the `batch-lines()` option if you modified it previously.

status_code	Complete response while running with trace messages enabled	Possible reason(s)	Possible solution (s)

	<pre>"status": "INVALID_ARGUMENT"
	}</pre>		

400	<pre>"error": {
	"code": 400,
	"message": "Request payload size exceeds the limit: 10485760 bytes.",
	"status": "INVALID_ARGUMENT"
	}</pre>
		The batch size is too large. Google Pub/Sub allows maximum 10MB per batch.	To overcome the issue, try one of the following methods:
			• Decrease the message size (consider the length of data and additional parameters combined).
			• Decrease the batch size with the `batch-lines()` option. Fewer, but bigger messages result in a smaller batch size, so adjust the value of the `batch-lines()` option to decrease your throughput.

403	<pre>"error": {
	"code": 403,
	"message": "User not authorized to perform this action.",
	"status": "PERMISSION_DENIED"
	}</pre>
		One of the following possible reasons behind the error message:
		• Wrong credentials.
		• Insufficient permissions.
			To overcome the issue, try one of the following methods:
			• Check your credentials `.JSON` file that you downloaded from the UI of

status_code	Complete response while running with trace messages enabled	Possible reason(s)	Possible solution (s)
			Google Pub/Sub. • Check the associated "roles" of your service account. The google_pubsub() destination requires the "Pub/Sub Publisher" role to operate.

404	"error": {
	"code": 404,
	"message": "Requested project not found or user does not have access to it (project=YOUR_PROJECT). Make sure to specify the unique project identifier and not the Google Cloud Console display name.",
	"status": "NOT_FOUND"
	}
		You have specified an incorrect project ID. The string YOUR_PROJECT is the project name provided in the configuration. and the project name you have to specify.	The project name you can find on the Pub/Sub UI is not necessarily the same as the project ID you specified in the YOUR_PROJECT string in your configuration. Make sure you use the project name provided in the YOUR_PROJECT string in your configuration.

404	"error": {
	"code": 404,
	"message": "Resource not found (resource=YOUR_TOPIC).",
	"status": "NOT_FOUND"
	}
		You have specified an incorrect topic ID. The string YOUR_TOPIC is the topic ID you provided in the configuration, and the topic ID you have to specify.	Make sure you use the topic ID you provided in the YOUR_TOPIC string in the configuration, and make sure that you have sufficient permissions to access it.

429	"error": { "code": 429, "message": "Quota	This error indicates that you have exceeded the quota for the given Google Cloud project.	Review your Google Cloud project's quota and adjust it according to

status_code	Complete response while running with trace messages enabled	Possible reason(s)	Possible solution (s)
	`exceeded for quota metric 'Regional publisher throughput, kB' and limit 'Regional publisher throughput, kB per minute per region' of service 'pubsubgoogleapiscom' for consumer 'project_number:127287437417'", "status": "RESOURCE_EXHAUSTED"}`		Google's documentation if necessary.

`hdfs`: Storing messages on the Hadoop Distributed File System (HDFS)

Starting with version 5.33.7, syslog-ng PE can send plain-text log files to the Hadoop Distributed File System (HDFS), allowing you to store your log data on a distributed, scalable file system. This is especially useful if you have huge amount of log messages that would be difficult to store otherwise, or if you want to process your messages using Hadoop tools (for example, Apache Pig).

NOTE: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Note the following limitations when using the syslog-ng PE`hdfs` destination:

- This destination is only supported on the Linux platform.

  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 HDFS client, the `hdfs` destination has significant memory usage (about 400MB).
- NOTE: You cannot set when log messages are flushed. Hadoop performs this action automatically, depending on its configured block size, and the amount of data received. There is no way for the syslog-ng PE application to influence when the messages are actually written to disk. This means that syslog-ng PE cannot guarantee that a message sent to HDFS is actually written to disk. When using flow-control, syslog-ng PE acknowledges a message as written to disk when it passes the message to the HDFS client. This method is as reliable as your HDFS environment.

- The log messages of the underlying client libraries are available in the `internal()` source of syslog-ng PE.

NOTE: The `hdfs` destination has been tested with Hortonworks Data Platform.

Declaration

```
@module  mod-java
 @include  "scl.conf"

 hdfs(
       client-lib-dir("/opt/syslog-ng/lib/syslog-ng/java-modules/:<path-to-
preinstalled-hadoop-libraries>")
       hdfs-uri("hdfs://NameNode:8020")
       hdfs-file("<path-to-logfile>")
 );
```

Example: Storing logfiles on HDFS

The following example defines an `hdfs` destination using only the required parameters.

```
@module  mod-java
 @include  "scl.conf"

 destination  d_hdfs  {
       hdfs(
             client-lib-dir("/opt/syslog-ng/lib/syslog-ng/java-
modules/:/opt/hadoop/libs")
             hdfs-uri("hdfs://10.140.32.80:8020")
             hdfs-file("/user/log/logfile.txt")
       );
 };
```

- To install the software required for the `hdfs` destination, see Prerequisites.
- For details on how the `hdfs` destination works, see How syslog-ng PE interacts with HDFS.
- For details on using MapR-FS, see Storing messages with MapR-FS.
- For the list of options, see HDFS destination options.

The `hdfs()` driver is actually a reusable configuration snippet configured to receive log messages using the Java language-binding of syslog-ng PE. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of the hdfs configuration snippet on GitHub. For details on extending syslog-ng PE in Java, see the Getting started with syslog-ng development guide.

NOTE: If you delete all Java destinations from your configuration and reload syslog-ng, the JVM is not used anymore, but it is still running. If you want to stop JVM, stop syslog-ng and then start syslog-ng again.

Prerequisites

The following describes how to send messages from syslog-ng PE to HDFS.

To send messages from syslog-ng PE to HDFS

1. If you want to use the Java-based modules of syslog-ng PE (for example, the Elasticsearch, HDFS, or Kafka destinations), download and install the Java Runtime Environment (JRE), 1.8.

   The Java-based modules of syslog-ng PE are tested and supported when using the Oracle implementation of Java. Other implementations are untested and unsupported, they may or may not work as expected.You can use OpenJDK or Oracle JDK, other implementations are not tested.
2. Download the Hadoop Distributed File System (HDFS) libraries (version 2.x) from http://hadoop.apache.org/releases.html.
3. Extract the HDFS libraries into a target directory (for example, `/opt/hadoop/lib/`), then execute the classpath command of the hadoop script: `bin/hdfs classpath`

   Use the classpath that this command returns in the syslog-ng PE configuration file, in the `client-lib-dir()` option of the HDFS destination.
4. Remove all `log4j` and `slf4j-log4j12` files from hadoop library, and download `log4j-slf4j-impl` (https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl/2.17.1) with matching version of `log4j2` found in syslog-ng premium edition. The `log4j-slf4j-impl` should be in one of the directories `bin/hdfs classpath` scripts returned.

How syslog-ng PE interacts with HDFS

The syslog-ng PE application sends the log messages to the official HDFS client library, which forwards the data to the HDFS nodes. The way how syslog-ng PE interacts with HDFS is described in the following steps.

1. After syslog-ng PE is started and the first message arrives to the `hdfs` destination, the `hdfs` destination tries to connect to the HDFS NameNode. If the connection fails, syslog-ng PE will repeatedly attempt to connect again after the period set in `time-reopen()` expires.
2. syslog-ng PE checks if the path to the logfile exists. If a directory does not exist syslog-ng PE automatically creates it. syslog-ng PE creates the destination file (using the filename set in the syslog-ng PE configuration file, with a UUID suffix to make it

unique, for example, `/usr/hadoop/logfile.txt.3dc1c59e-ab3b-4b71-9e81-93db477ed9d9`) and writes the message into the file. After the file is created, syslog-ng PE will write all incoming messages into the `hdfs` destination.

> NOTE: When the `hdfs-append-enabled()` option is set to `true`, syslog-ng PE will not assign a new UUID suffix to an existing file, because it is then possible to open a closed file and append data to that.

> NOTE: You cannot set when log messages are flushed. Hadoop performs this action automatically, depending on its configured block size, and the amount of data received. There is no way for the syslog-ng PE application to influence when the messages are actually written to disk. This means that syslog-ng PE cannot guarantee that a message sent to HDFS is actually written to disk. When using flow-control, syslog-ng PE acknowledges a message as written to disk when it passes the message to the HDFS client. This method is as reliable as your HDFS environment.

3. If the HDFS client returns an error, syslog-ng PE attempts to close the file, then opens a new file and repeats sending the message (trying to connect to HDFS and send the message), as set in the `retries()` parameter. If sending the message fails for `retries()` times, syslog-ng PE drops the message.
4. The syslog-ng PE application closes the destination file in the following cases:
   - syslog-ng PE is reloaded
   - syslog-ng PE is restarted
   - The HDFS client returns an error.
5. If the file is closed and you have set an archive directory, syslog-ng PE moves the file to this directory. If syslog-ng PE cannot move the file for some reason (for example, syslog-ng PE cannot connect to the HDFS NameNode), the file remains at its original location, syslog-ng PE will not try to move it again.

Storing messages with MapR-FS

The syslog-ng PE application is also compatible with MapR File System (MapR-FS), starting from version 5.4, syslog-ng Premium Edition is *MapR* certified. MapR-FS provides better performance, reliability, efficiency, maintainability, and ease of use compared to the default Hadoop Distributed Files System (HDFS). To use MapR-FS with syslog-ng PE, complete the following steps:

1. Install MapR libraries. Instead of the official Apache HDFS libraries, MapR uses different libraries. The supported version is MapR 4.x.
   a. Download the libraries from the Maven Repository and Artifacts for MapR or get it from an already existing MapR installation.
   b. Install MapR. If you do not know how to install MapR, follow the instructions on the MapR website.
2. In a default MapR installation, the required libraries are installed in the following path: `/opt/mapr/lib`.

Enter the path where MapR was installed in the `class-path` option of the `hdfs` destination, for example:

```
class-path("/opt/mapr/lib/")
```

If the libraries were downloaded from the Maven Repository, the following additional libraries will be requiered. Note that the version numbers in the filenames can be different in the various Hadoop releases:`commons-collections-3.2.1.jar`, `commons-logging-1.1.3.jar`, `hadoop-auth-2.5.1.jar`, `log4j-1.2.15.jar`, `slf4j-api-1.7.5.jar`, `commons-configuration-1.6.jar`, `guava-13.0.1.jar`, `hadoop-common-2.5.1.jar`, `maprfs-4.0.2-mapr.jar`, `slf4j-log4j12-1.7.5.jar`, `commons-lang-2.5.jar`, `hadoop-0.20.2-dev-core.jar`, `json-20080701.jar`, `protobuf-java-2.5.0.jar`, `zookeeper-3.4.5-mapr-1406.jar`.

3. Configure the `hdfs` destination in syslog-ng PE.

Example: Storing logfiles with MapR-FS

The following example defines an `hdfs` destination for MapR-FS using only the required parameters.

```
@module  mod-java
 @include  "scl.conf"

 destination  d_mapr  {
       hdfs(
             client-lib-dir("/opt/syslog-ng/lib/syslog-ng/java-
modules/:/opt/mapr/lib/")
             hdfs-uri("maprfs://10.140.32.80")
             hdfs-file("/user/log/logfile.txt")
       );
 };
```

Kerberos authentication with syslog-ng hdfs () destination

Version 3.107.0.3 and later supports Kerberos authentication to authenticate the connection to your Hadoop cluster. syslog-ng PE assumes that you already have a Hadoop and Kerberos infrastructure.

NOTE: If you configure Kerberos authentication for a `hdfs()` destination, it affects all `hdfs()` destinations. Kerberos and non-Kerberos `hdfs()` destinations cannot be mixed in a syslog-ng PE configuration. This means that if one `hdfs()` destination uses Kerberos authentication, you have to configure all other `hdfs()` destinations to use Kerberos

authentication too.

Failing to do so results in non-Kerberos `hdfs()` destinations being unable to authenticate to the HDFS server.

NOTE: If you want to configure your `hdfs()` destination to stop using Kerberos authentication, namely, to remove Kerberos-related options from the `hdfs()` destination configuration, make sure to restart syslog-ng PE for the changes to take effect.

Prerequisites

- You have configured your Hadoop infrastructure to use Kerberos authentication.
- You have a keytab file and a principal for the host running syslog-ng PE.
- You have installed and configured the Kerberos client packages on the host running syslog-ng PE. (That is, Kerberos authentication works for the host, for example, from the command line using the `kinit user@REALM -k -t <keytab_file>` command.)

```
destination d_hdfs {
       hdfs(
              client-lib-dir("/hdfs-libs/lib")
              hdfs-uri("hdfs://hdp-kerberos.syslog-ng.example:8020")
              kerberos-keytab-file("/opt/syslog-ng/etc/hdfs.headless.keytab")
              kerberos-principal("hdfs-hdpkerberos@MYREALM")
              hdfs-file("/var/hdfs/test.log")
       );
};
```

HDFS destination options

The `hdfs` destination stores the log messages in files on the Hadoop Distributed File System (HDFS). The `hdfs` destination has the following options.

The following options are required: `hdfs-file()`, `hdfs-uri()`. Note that to use `hdfs`, 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:	The syslog-ng PE module directory: /opt/syslog-ng/lib/syslog-ng/java-modules/

Description: The list of the paths where the required Java classes are located. For example, `class-path("/opt/syslog-ng/lib/syslog-ng/java-modules/:/opt/my-java-`

`libraries/libs/")`. If you set this option multiple times in your syslog-ng PE configuration (for example, because you have multiple Java-based destinations), syslog-ng PE will merge every available paths to a single list.

For the `hdfs` destination, include the path to the directory where you copied the required libraries (see Prerequisites), for example, `client-lib-dir("/opt/syslog-ng/lib/syslog-ng/java-modules/:/opt/hadoop/libs/")`.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

hdfs-append-enabled()

Type:	true | false
Default:	false

Description: When `hdfs-append-enabled` is set to `true`, syslog-ng PE will append new data to the end of an already existing HDFS file. Note that in this case, archiving is automatically disabled, and syslog-ng PE will ignore the `hdfs-archive-dir` option.

When `hdfs-append-enabled` is set to `false`, the syslog-ng PE application always creates a new file if the previous has been closed. In that case, appending data to existing files is not supported.

When you choose to write data into an existing file, syslog-ng PE does not extend the filename with a UUID suffix because there is no need to open a new file (a new unique ID would mean opening a new file and writing data into that).

CAUTION:

Before enabling the `hdfs-append-enabled` option, ensure that your HDFS server supports the `append` operation and that it is enabled. Otherwise syslog-ng PE will not be able to append data into an existing file, resulting in an error log.

hdfs-archive-dir()

Type:	string
Default:	N/A

Description: The path where syslog-ng PE will move the closed log files. If syslog-ng PE cannot move the file for some reason (for example, syslog-ng PE cannot connect to the HDFS NameNode), the file remains at its original location. For example, `hdfs-archive-dir ("/usr/hdfs/archive/")`.

NOTE: When `hdfs-append-enabled` is set to `true`, archiving is automatically disabled, and syslog-ng PE will ignore the `hdfs-archive-dir` option.

hdfs-file()

Type:	string
Default:	N/A

Description: The path and name of the log file. For example, `hdfs-file ("/usr/hdfs/mylogfile.txt")`. syslog-ng PE checks if the path to the logfile exists. If a directory does not exist syslog-ng PE automatically creates it.

`hdfs-file()` supports the usage of macros. This means that syslog-ng PE can create files on HDFS dynamically, using macros in the file (or directory) name.

NOTE: When a filename resolved from the macros contains a character that HDFS does not support, syslog-ng PE will not be able to create the file. Make sure that you use macros that do not contain unsupported characters.

Example: Using macros in filenames

In the following example, a `/var/testdb_working_dir/$DAY-$HOUR.txt` file will be created (with a UUID suffix):

```
destination  d_hdfs_9bf3ff45341643c69bf46bfff940372a  {
       hdfs(
              client-lib-dir(/hdfs-libs)
              hdfs-uri("hdfs://hdp2.syslog-ng.example:8020")
              hdfs_file("/var/testdb_working_dir/$DAY-$HOUR.txt")
       );
 };
```

As an example, if it is the 31st day of the month and it is 12 o'clock, then the name of the file will be `31-12.txt`.

hdfs-max-filename-length()

Type:	number
Default:	255

Description: The maximum length of the filename. This filename (including the UUID that syslog-ng PE appends to it) cannot be longer than what the file system permits. If the filename is longer than the value of `hdfs-max-filename-length`, syslog-ng PE will automatically truncate the filename. For example, `hdfs-max-filename-length("255")`.

hdfs-resources()

Type:	string
Default:	N/A

Description: The list of Hadoop resources to load, separated by semicolons. For example, `hdfs-resources("/home/user/hadoop/core-site.xml;/home/user/hadoop/hdfs-site.xml")`.

hdfs-uri()

Type:	string
Default:	N/A

Description: The URI of the HDFS NameNode is in `hdfs://IPaddress:port` or `hdfs://hostname:port` format. When using MapR-FS, the URI of the MapR-FS NameNode is in `maprfs://IPaddress` or `maprfs://hostname` format, for example: `maprfs://10.140.32.80`. The IP address of the node can be IPv4 or IPv6. For example, `hdfs-uri ("hdfs://10.140.32.80:8020")`. The IPv6 address must be enclosed in square brackets (*[]*) as specified by RFC 2732, for example, `hdfs-uri("hdfs:// [FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]:8020")`.

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng PE configuration file.

For example:

```
jvm-options("-Xss1M -XX:+TraceClassLoading")
```

You can set this option only as a global option, by adding it to the `options` statement of the syslog-ng configuration file.

kerberos-keytab-file()

Type:	string
Default:	N/A

Description: The path to the Kerberos keytab file that you received from your Kerberos administrator. For example, `kerberos-keytab-file("/opt/syslog-ng/etc/hdfs.headless.keytab")`. This option is needed only if you want to authenticate using Kerberos in Hadoop. You also have to set the `hdfs-option-kerberos-principal()`

option. For details on the using Kerberos authentication with the `hdfs()` destination, see Kerberos authentication with syslog-ng hdfs() destination.

```
destination d_hdfs {
       hdfs(
              client-lib-dir("/hdfs-libs/lib")
              hdfs-uri("hdfs://hdp-kerberos.syslog-ng.example:8020")
              kerberos-keytab-file("/opt/syslog-ng/etc/hdfs.headless.keytab")
              kerberos-principal("hdfs-hdpkerberos@MYREALM")
              hdfs-file("/var/hdfs/test.log")
       );
};
```

Available in syslog-ng PE version 3.107.0.3 and later.

kerberos-principal()

Type:	string
Default:	N/A

Description: The Kerberos principal you want to authenticate with. For example, `kerberos-principal("hdfs-user@MYREALM")`. This option is needed only if you want to authenticate using Kerberos in Hadoop. You also have to set the `hdfs-option-kerberos-keytab-file()` option. For details on the using Kerberos authentication with the `hdfs()` destination, see Kerberos authentication with syslog-ng hdfs() destination.

```
destination d_hdfs {
       hdfs(
              client-lib-dir("/hdfs-libs/lib")
              hdfs-uri("hdfs://hdp-kerberos.syslog-ng.example:8020")
              kerberos-keytab-file("/opt/syslog-ng/etc/hdfs.headless.keytab")
              kerberos-principal("hdfs-hdpkerberos@MYREALM")
              hdfs-file("/var/hdfs/test.log")
       );
};
```

Available in syslog-ng PE version 3.107.0.3 and later.

log-fifo-size()

Type:	number
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.

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.

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reap()

Accepted values:	number (seconds)
Default:	0 (disabled)

Description: The time to wait in seconds before an idle destination file is closed. Note that if `hdfs-archive-dir` option is set and `time-reap` expires, archiving is triggered for the affected file.

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 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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

`http`: Posting messages over HTTP without Java

Version 3.87.0.4 of syslog-ng PE can directly post log messages to web services using the HTTP protocol, without having to use Java.

Limitations

The current implementation of the `http()` destination has the following limitations:

- Only the PUT and the POST methods are supported.
- HTTPS connections, as well as password-based and certificate-based authentication, are supported.
- If the server returns a status code beginning with 4 (for example, 404) to the POST or PUT request, syslog-ng PE drops the message without attempting to resend it.

NOTE: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Example: Client certificate authentication with HTTPS

```
destination  d_https  {
       http(
              [...]
              tls(
                    ca-file("/<path-to-certificate-directory>/ca-crt.pem")
                    ca-dir("/<path-to-certificate-directory>/")
                    cert-file("/<path-to-certificate-directory>/server-
crt.pem")
                    key-file("/<path-to-certificate-directory>/server-
key.pem")
                    )
              [...]
       );
 };
```

Declaration

```
destination d_http {
    http(
        url("<web-service-IP-or-hostname>")
        method("<HTTP-method>")
        user-agent("<USER-AGENT-message-value>")
```

```
        user("<username>")
        password("<password>")
    );
};
```

Example: Sending log data to a web service

The following example defines an `http()` destination.

```
destination d_http {
       http(
             url("http://127.0.0.1:8000")
             method("PUT")
             user-agent("syslog-ng User Agent")
             user("user")
             password("password")
             headers("HEADER1: header1", "HEADER2: header2")
             body("${ISODATE} ${MESSAGE}")
       );
};

log {
       source(s_file);
       destination(d_http);
       flags(flow-control);
};
```

Batch mode and load balancing

Starting with version 3.187.0.12, you can send multiple log messages in a single HTTP request if the destination HTTP server supports that.

Batch size

The `batch-lines()`, `batch-lines()`, and `batch-timeout()` options of the destination determine how many log messages syslog-ng PE sends in a batch. The `batch-lines()` option determines the maximum number of messages syslog-ng PE puts in a batch in. This can be limited based on size and time:

- syslog-ng PE sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. This ensures that the destination receives every message in a timely manner even if suddenly there are no

more messages.

- syslog-ng PE sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Formatting the batch

By default, syslog-ng PE separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option.

If the target application or server requires a special beginning or ending to recognize batches, use the `body-prefix()` and `body-suffix()` options to add a beginning and ending to the batch. For example, you can use these options to create JSON-encoded arrays as POST payloads, which is required by a number of REST APIs. The body of a batch HTTP request looks like this:

```
value of body-prefix() option
log-line-1 (as formatted in the body() option)
log-line-2 (as formatted in the body() option)
....
log-line-n (the number of log lines is batch-lines(), or less if batch-timeout()
has elapsed or the batch would be longer than batch-bytes())
value of body-suffix() option
```

Example: HTTP batch mode

The following destination sends log messages to an Elasticsearch server using the bulk API. A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 10 seconds (10000 milliseconds).

```
destination d_http {
    http(url("http://your-elasticsearch-server/_bulk")
        method("POST")
        batch-lines(100)
        batch-bytes(512Kb)
        batch-timeout(10000)
        headers("Content-Type: application/x-ndjson")
```

```
        body-suffix("\n")
        body('{ "index":{} }
             $(format-json --scope rfc5424 --key ISODATE)')
    );
};
```

Load balancing between multiple servers

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

Example: HTTP load balancing

The following destination sends log messages to an Elasticsearch server using the bulk API, to 3 different ingest nodes. Each node is assigned a separate worker thread. A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 10 seconds (10000 milliseconds).

```
destination d_http {
    http(url("http://your-elasticsearch-server/_bulk" "http://your-second-
ingest-node/_bulk" "http://your-third-ingest-node/_bulk")
        method("POST")
        batch-lines(100)
        batch-bytes(512Kb)
        batch-timeout(10000)
        workers(3)
        headers("Content-Type: application/x-ndjson")
        body-suffix("\n")
```

```
        body('{ "index":{} }
             $(format-json --scope rfc5424 --key ISODATE)')
        persist-name("d_http-load-balance")
    );
};
```

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

HTTP destination options

The `http` destination of syslog-ng PE can directly post log messages to web services using the HTTP protocol. The `http` destination has the following options.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, syslog-ng PE sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in syslog-ng PE version 3.197.0.12 and later.

For details on how this option influences HTTP batch mode, see http: Posting messages over HTTP without Java on page 413

batch-lines()

Type:	number [lines]
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

For details on how this option influences HTTP batch mode, see http: Posting messages over HTTP without Java on page 413

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

For details on how this option influences HTTP batch mode, see http: Posting messages over HTTP without Java on page 413

body()

Type:	string or template
Default:	

Description: The body of the HTTP request, for example, `body("${ISODATE} ${MESSAGE}")`. You can use strings, macros, and template functions in the body. If not set, it will contain the message received from the source by default.

body-prefix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts at the beginning of the body of the HTTP request, before the log message. Available in syslog-ng PE version 3.187.0.11 and later.

For details on how this option influences HTTP batch mode, see http: Posting messages over HTTP without Java on page 413

body-suffix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts to the end of the body of the HTTP request, after the log message. Available in syslog-ng PE version 3.187.0.11 and later.

For details on how this option influences HTTP batch mode, see http: Posting messages over HTTP without Java on page 413

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. For an example, see Configuring TLS on the syslog-ng clients. The syslog-ng PE application uses the CA certificates in this directory to validate the certificate of the peer.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
       http(
               url("http://127.0.0.1:8080")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
```

```
                    cipher-suite("cipher")
                    key-file("key")
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the HTTPS server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 set in the `key-file()` option. The syslog-ng PE application uses this certificate to authenticate the syslog-ng PE client on the destination server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that syslog-ng PE uses

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")
```

For a list of available algorithms, execute the `openssl ciphers -v` command. The first column of the output contains the name of the algorithms to use in the `cipher-suite()` option, the second column specifies which encryption protocol uses the algorithm (for example, `TLSv1.2`). That way, the `cipher-suite()` also determines the encryption protocol used in the connection: to disable SSLv3, use an algorithm that is available only in TLSv1.2, and that both the client and the server supports. You can also specify the encryption protocols using ssl-options().

You can also use the following command to automatically list only ciphers permitted in a specific encryption protocol, for example, `TLSv1.2`:

```
echo "cipher-suite(\"$(openssl ciphers -v | grep TLSv1.2 | awk '{print $1}' |
xargs echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, syslog-ng PE separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option. Available in syslog-ng PE version 3.187.0.11 and later.

For details on how this option influences HTTP batch mode, see http: Posting messages over HTTP without Java on page 413

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file

can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
              )
       );
};
```

headers()

Type:	string list
Default:	

Description: Custom HTTP headers to include in the request, for example, `headers ("HEADER1: header1", "HEADER2: header2")`. If not set, only the default headers are included, but no custom headers.

The following headers are included by default:

- X-Syslog-Host: <host>
- X-Syslog-Program: <program>
- X-Syslog-Facility: <facility>
- X-Syslog-Level: <loglevel/priority>

hook-commands()

Description: This option makes it possible to execute external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to execute external applications.

Using the hook-commands() when syslog-ng PE starts or stops

To execute an external program when syslog-ng PE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed when syslog-ng PE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed when syslog-ng PE stops.	

Using the hook-commands() when syslog-ng PE reloads

To execute an external program when the syslog-ng PE configuration is initiated or torn down (for example, on startup/shutdown or during a syslog-ng PE reload), use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.

Example: Using the `hook-commands()` with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically when syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow (otherwise the port is closed).

```
source  {
    network(transport(udp)
       hook-commands(
          startup("iptables  -I  LOGCHAIN  1  -p  udp  --dport  514  -j
ACCEPT")
          shutdown("iptables  -D  LOGCHAIN  1")
       ;)
    );
 };
```

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

key-file()

Accepted values:	Filename
Default:	none

Description: Path and name of a file that contains a private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng PE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng PE client on the destination server.

The `http()` destination supports only unencrypted key files (that is, the private key cannot be password-protected).

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

method()

Type:	POST | PUT
Default:	POST

Description: Specifies the HTTP method to use when sending the message to the server.

password()

Type:	string
Default:	

Description: The password that syslog-ng PE uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	yes | no
Default:	yes

Description: Verification method of the peer. 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	*no (optional-untrusted)*	TLS-encryption	TLS-encryption	TLS-encryption
	yes (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.

persist-name()

Type:	string
Default:	None

*Description:*If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

The `proxy()` option enables you to configure the `HTTP` driver to use a specific HTTP proxy for all HTTP-based destinations, instead of using the proxy that is configured for the system.

If you do not set the `proxy()` option, the HTTP driver uses the `http_proxy` and `https_proxy` environment variables, as shown in CURLOPT_PROXY explained.

NOTE: Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.

Example: the `proxy()` option in configuration

The following example illustrates including the `proxy()` option in your configuration.

```
destination {
                                http( url("SYSLOG_SERVER_IP:PORT") proxy
("PROXY_IP:PORT") method("POST"));
                          };
```

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.

To handle HTTP error responses, if the HTTP server returns 5xx codes, syslog-ng PE will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, syslog-ng PE will drop the messages.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

timeout()

Type:	number [seconds]
Default:	0

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded. By default, the timeout value is `0`, meaning that there is no timeout. Available in version 3.117.0.4 and later.

url()

Type:	URL or list of URLs
Default:	http://localhost/

Description: Specifies the hostname or IP address, and optionally the port number of the web service that can receive log data through HTTP. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://127.0.0.1:8000`

In case the server on the specified URL returns a redirect request, syslog-ng PE automatically follows maximum 3 redirects. Only HTTP-based and HTTPS-based redirections are supported.

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

user-agent()

Type:	string
Default:	syslog-ng [version]/libcurl[version]

Description: The value of the USER-AGENT header in the messages sent to the server.

user()

Type:	string
Default:	

Description: The username that syslog-ng PE uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	yes | no
Default:	no

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

kafka(): Publishing messages to Apache Kafka (Java implementation) (DEPRECATED)

Starting with version 5.43.7, syslog-ng PE can directly publish log messages to the Apache Kafka message bus, where subscribers can access them.

NOTE: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

NOTE: From syslog-ng PE version 7.0.26, the old, Java-based `kafka()` destination has been deprecated. One Identity recommends that you use the new, C-based `kafka-c()` destination.

- This destination is only supported on the Linux platform.

  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 Kafka producer, the `kafka()` 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"

 kafka(
       client-lib-dir("/opt/syslog-ng/lib/syslog-ng/java-modules/:<path-to-
preinstalled-kafka-libraries>")
       kafka-bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
       topic("${HOST}")
 );
```

Example: Sending log data to Apache Kafka

The following example defines a `kafka()` destination, using only the required parameters.

```
@module  mod-java
 @include  "scl.conf"

 destination  d_kafka  {
       kafka(
             client-lib-dir(/opt/syslog-ng/lib/syslog-ng/java-
modules/KafkaDestination.jar:/usr/share/kafka/lib/)
             kafka-bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
             topic("${HOST}")
       );
 };
```

- To install the software required for the `kafka()` destination, see Prerequisites.
- For details on how the `kafka()` destination works, see How syslog-ng PE interacts with Apache Kafka.
- For the list of options, see Kafka destination options.

The `kafka()` driver is actually a reusable configuration snippet configured to receive log messages using the Java language-binding of syslog-ng PE. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of the kafka configuration snippet on GitHub. For details on extending syslog-ng PE in Java, see the Getting started with syslog-ng PE development guide.

NOTE: If you delete all Java destinations from your configuration and reload syslog-ng, the JVM is not used anymore, but it is still running. If you want to stop JVM, stop syslog-ng and then start syslog-ng again.

Prerequisites

This section describes how to publish messages from syslog-ng PE to Apache Kafka.

To publish messages from syslog-ng PE to Apache Kafka

1. If you want to use the Java-based modules of syslog-ng PE (for example, the Elasticsearch, HDFS, or Kafka destinations), download and install the Java Runtime Environment (JRE), 1.8.

   The Java-based modules of syslog-ng PE are tested and supported when using the Oracle implementation of Java. Other implementations are untested and unsupported, they may or may not work as expected.You can use OpenJDK or Oracle JDK, other implementations are not tested.
2. Download the latest stable binary release of the Apache Kafka libraries (version 0.9 or newer) from http://kafka.apache.org/downloads.html.

3. Extract the Apache Kafka libraries into a single directory. If needed, collect the various `.jar` files into a single directory (for example, `/opt/kafka/lib/`) where syslog-ng PE can access them. You must specify this directory in the syslog-ng PE configuration file.
4. Remove all `log4j` and `slf4j-log4j12` files from hadoop library, and download `log4j-slf4j-impl` (https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl/2.17.1) with matching version of `log4j2` found in syslog-ng premium edition. The `log4j-slf4j-impl` should be in one of the directories `bin/hdfs classpath` scripts returned.

How syslog-ng PE interacts with Apache Kafka

When stopping the syslog-ng PE application, syslog-ng PE will not stop until all Java threads are finished, including the threads started by the Kafka Producer. There is no way (except for the `kill -9` command) to stop syslog-ng PE before the Kafka Producer stops. To change this behavior set the properties of the Kafka Producer in its properties file, and reference the file in the `properties-file` option.

The syslog-ng PE `kafka()` destination tries to reconnect to the brokers in a tight loop. This can look as spinning, because of a lot of similar debug messages. To decrease the amount of such messages, set a bigger timeout using the following properties:

```
retry.backoff.ms=1000
reconnect.backoff.ms=1000
```

For details on using property files, see properties-file(). For details on the properties that you can set in the property file, see the Apache Kafka documentation.

Kafka destination options

The `kafka()` destination of syslog-ng PE can directly publish log messages to the Apache Kafka message bus, where subscribers can access them. The `kafka()` destination has the following options.

Required options

The following options are required: `kafka-bootstrap-servers()`, `topic()`. Note that to use the `kafka()` destination, 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:	The syslog-ng PE module directory: /opt/syslog-ng/lib/syslog-ng/java-modules/

Description: The list of the paths where the required Java classes are located. For example, `class-path("/opt/syslog-ng/lib/syslog-ng/java-modules/:/opt/my-java-libraries/libs/")`. If you set this option multiple times in your syslog-ng PE configuration (for example, because you have multiple Java-based destinations), syslog-ng PE will merge every available paths to a single list.

For the `kafka()` destination, include the path to the directory where you copied the required libraries (see Prerequisites), for example, `client-lib-dir(/opt/syslog-ng/lib/syslog-ng/java-modules/KafkaDestination.jar:/usr/share/kafka/lib/*.jar)`.

kafka-bootstrap-servers()

Type:	list of hostnames
Default:	

Description: Specifies the hostname or IP address of the Kafka server. When specifying an IP address, IPv4 (for example, `192.168.0.1`) or IPv6 (for example, `[::1]`) can be used as well. Use a colon (`:`) after the address to specify the port number of the server. When specifying multiple addresses, use a comma to separate the addresses, for example, `kafka-bootstrap-servers("127.0.0.1:2525,remote-server-hostname:6464")`

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

> NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng PE configuration file.

For example:

```
jvm-options("-Xss1M -XX:+TraceClassLoading")
```

You can set this option only as a global option, by adding it to the `options` statement of the syslog-ng configuration file.

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.

key()

Type:	template
Default:	N/A

Description: The key of the partition under which the message is published. You can use templates to change the topic dynamically based on the source or the content of the message, for example, `key("${PROGRAM}")`.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

properties-file()

Type:	string (absolute path)
Default:	N/A

Description: The absolute path and filename of the Kafka properties file to load. For example, `properties-file("/opt/syslog-ng/etc/kafka_dest.properties")`. The syslog-ng PE application reads this file and passes the properties to the Kafka Producer. If a property is defined both in the syslog-ng PE configuration file (`syslog-ng.conf`) and in the properties file, then syslog-ng PE uses the definition from the syslog-ng PE configuration file.

The syslog-ng PE `kafka()` destination supports all properties of the official Kafka producer. For details, see the Apache Kafka documentation.

The `kafka-bootstrap-servers` option is translated to the `bootstrap.servers` property.

For example, the following properties file defines the acknowledgment method and compression:

```
acks=all
compression.type=snappy
```

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.

sync-send()

Type:	true | false
Default:	false

Description: When `sync-send` is set to `true`, syslog-ng PE sends the message reliably: it sends a message to the Kafka server, then waits for a reply. 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 very slow.

When `sync-send` is set to `false`, syslog-ng PE sends messages asynchronously, and receives the response asynchronously. In case of a problem, syslog-ng PE cannot resend the messages.

This method is fast, but the transfer is not reliable. Several thousands of messages can be lost before syslog-ng PE recognizes the error.

template()

Type:	template or template function
Default:	$ISODATE $HOST $MSGHDR$MSG\n

Description: The message as published to Apache Kafka. You can use templates and template functions (for example, `format-json()`) to format the message, for example, `template("$(format-json --scope rfc5424 --exclude DATE --key ISODATE)")`.

For details on formatting messages in JSON format, see format-json.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

topic()

Type:	template
Default:	N/A

Description: The Kafka topic under which the message is published. You can use templates to change the topic dynamically based on the source or the content of the message, for example, `topic("${HOST}")`.

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 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:	rfc3164

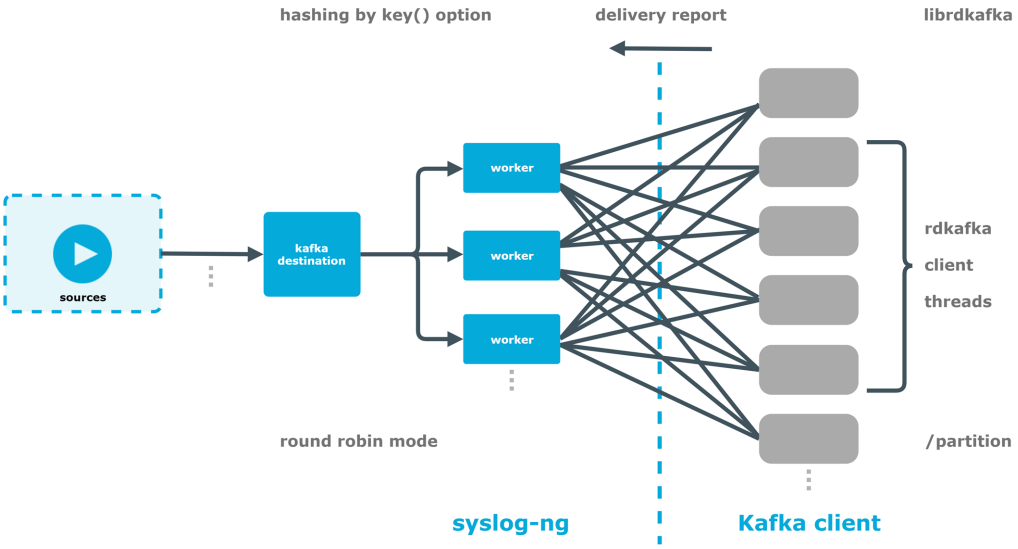
Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

kafka-c(): Publishing messages to Apache Kafka using the librdkafka client (C implementation)

As of syslog-ng PE version 7.0.26, the `kafka-c()` destination can directly publish log messages to the Apache Kafka message bus using the librdkafka client. The new, C-based implementation has several advantages in comparison with the Java-based implementation, such as scalability, more efficient memory usage, and simpler setup.

The following figure illustrates how the `kafka-c()` destination works with syslog-ng PE.

Figure 28: How the kafka-c() destination works with syslog-ng PE



kafka-c(): Prerequisites and limitations

This section describes the prerequisites and restrictions for using the `kafka-c()` destination, and important information about the declaring the destination.

Prerequisites and restrictions

- Since the new, C-based implementation uses the librdkafka client library, the `kafka-c()` destination has less memory usage than the previous, Java-based implementation (which uses the official Java Kafka producer).
- The log messages of the underlying client libraries are available in the `internal()` source of syslog-ng PE.
- If you used the Java implementation before, see kafka-c(): Shifting from the Java implementation to the C implementation .
- The syslog-ng PE `kafka-c()` destination supports all properties of the official Kafka producer. For details, see the librdkafka documentation.
- For the list of options, see Options of the kafka-c() destination.

Declaration

```
kafka-c(
   bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
   topic("topic-name")
);
```

Example: Sending log data to Apache Kafka

The following example defines a `kafka-c()` destination in the new C implementation, using only the required parameters.

```
@include "scl.conf"

   destination d_kafka {
      kafka-c(
         bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
         topic("topic-name")
      );
   };
```

kafka-c(): Shifting from the Java implementation to the C implementation

If you were using the Java-based `kafka()` destination and want to use the C-based `kafka-c()` destination, the following changes to the configuration file and considerations are necessary.

- The `client_lib_dir()` option has been deprecated. Remove it from the configuration file.
- The `kafka-bootstrap-servers()` option has been renamed as `bootstrap-servers()`.
- The `option()` option has been removed and replaced by the `config()` option, which has a different syntax.
- Instead of the `properties-file()` option, you can use the `config()` option (using a `config(key => value)` format) to fine-tune your configuration.

  NOTE: If you used the `properties-file()` option before, you can import the configuration parameters you were using earlier, with minor modifications in syntax into the `config()` option.

  Syntactical differences between the properties-file() option and the config() option

  The following examples illustrate the syntactical differences of using configuration parameters in the `properties-file()` option and the `config()` option:

- **properties-file()**

  ```
  content of "file.properties":
  acks=all
  compression.type=snappy
  ```

- **config()**

  ```
  config  (
     "acks"  =>  "all"
     "compression.type"  =>  "snappy"
  )
  ```

- The `template()` option has been renamed as `message()`.
- If you use templates with the `topic()` option, configuring the `fallback-topic()` option is also required.

For more information about these options, see Options of the kafka-c() destination.

kafka-c(): Flow control in syslog-ng PE and the Kafka client

A syslog-ng Premium Edition (syslog-ng PE) destination recognizes a message as sent when the message has been sent to the Kafka client, not when the Kafka server confirms its delivery.

If the Kafka client collects too many unsent messages, it will not accept any more messages from syslog-ng PE. The syslog-ng PE application detects this and stops sending messages to the Kafka client. Also, syslog-ng PE's flow control starts functioning in the direction of the sources (for example, syslog-ng PE will not read from the sources in that specific logpath).

CAUTION:

Hazard of data loss!

If `sync-send()` is set to `"no"`, the messages passed to the Kafka client can be lost.

To avoid data loss, One Identity recommends that you set `sync-send()` to `"yes"`, as this setting delivers messages to the Kafka client more reliably.

For more information, see the description of the `sync-send()` option.

Options of the kafka-c() destination

With the `kafka-c()` destination of syslog-ng PE, you can directly publish log messages to the Apache Kafka message bus, where subscribers can access them. The `kafka-c()` destination has the following options.

Required options

The following options are required:

- `bootstrap-servers()`
- `topic()`.

batch-lines()

Type:	number [lines]
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

NOTE: The syslog-ng PE configuration accepts this option with `sync-send()` set to both `"yes"` or `"no"`, but the option will only take effect if you set `sync-send()` to `"yes"`.

NOTE: If you set `sync-send()` to `"yes"`, the number you specify for `batch-lines()` affects how many messages syslog-ng PE packs into once transaction.

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

NOTE: The syslog-ng PE configuration accepts this option with `sync-send()` set to both `"yes"` or `"no"`, but the option will only take effect if you set `sync-send()` to `"yes"`.

NOTE: When setting `batch-timeout()`, consider the value of the `transaction.timeout.ms` Kafka property. If in case of timeout (that is, if syslog-ng PE does not receive `batch-lines()` amount of messages) the value of `batch-timeout()` exceeds the value of `transaction.timeout.ms`, syslog-ng PE will not send out messages in time.

For more information about the default values of the `transaction.timeout.ms` Kafka property, see the librdkafka documentation.

bootstrap-servers()

Type:	string
Default:	N/A

Description: Required option. Specifies the hostname or IP address of the Kafka server. When specifying an IP address, IPv4 (for example, `192.168.0.1`) or IPv6 (for example, `[::1]`) can be used as well. Use a colon (`:`) after the address to specify the port number of the server. When specifying multiple addresses, use a comma to separate the addresses, for example, `bootstrap-servers("127.0.0.1:2525,remote-server-hostname:6464")`

config()

Type:	N/A
Default:	N/A

Description: Advanced configuration option to fine-tune all properties of the official Kafka producer. For details, see the librdkafka documentation.

The syntax of the `config()` option is the following:

```
config (
    "acks" => "all"
    "compression.type" => "snappy"
)
```

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 more 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.

compaction()	
Type:	yes|no
Default:	no

Description: If set to `yes`, syslog-ng PE prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

CAUTION:

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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`.	

qout-size()	
Type:	number (messages)
Default:	64
Description: The number of messages stored in the output buffer of the destination.	

> NOTE: If you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `disk-buf-size()` are required options.

Example: 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")
              )
       );
 };
```

fallback-topic()

Type:	string
Default:	N/A

Description: If the resolved `topic()` template is not a valid Kafka topic , syslog-ng PE will use `fallback-topic()` to send messages.

NOTE: If instead of strings, you use actual templates (that is, a macro like `${MESSAGE}`, or a template function like `$(format-json)`) in the `topic()` option, configuring the `fallback-topic()` option is required.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

NOTE: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

flush-timeout-on-reload()

Type:	integer in **`milliseconds`**
Default:	1000

Description: When syslog-ng PE reloads, the Kafka client will also reload.

The `flush-timeout-on-reload()` option specifies the number of milliseconds syslog-ng PE waits for the Kafka client to flush out in-flight messages. In-flight messages may be:

- messages that are passed to the Kafka client for sending, which have been sent, but not delivered
- messages not yet sent out.

flush-timeout-on-shutdown()

Type:	integer in **`milliseconds`**
Default:	60000

Description: When syslog-ng PE shuts down, the Kafka client will also shut down.

The `flush-timeout-on-shutdown()` option specifies the number of milliseconds syslog-ng PE waits for the Kafka client to flush out in-flight messages. In-flight messages may be:

- Messages passed to the Kafka client for sending, already sent, but not yet delivered.
- Messages not yet sent by the Kafka client.

NOTE: To avoid losing messages, One Identity recommends that you use the `sync-send()` option set to `"yes"` in addition to using the `disk-buffer()` option.

hook-commands()

Description: This option makes it possible to run external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to run external applications.

Using hook-commands() when syslog-ng PE starts or stops

To run an external program when syslog-ng PE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is run as syslog-ng PE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is run as syslog-ng PE stops.	

Using hook-commands() when syslog-ng PE reloads

To run an external program when the syslog-ng PE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng PE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is run when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.	

teardown()	
Type:	string
Default:	N/A

Description: Defines an external program that is run when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow, otherwise the port is closed.

```
source  {
     network(transport(udp)
       hook-commands(
          startup("iptables  -I  LOGCHAIN  1  -p  udp  --dport  514  -j
ACCEPT")
          shutdown("iptables  -D  LOGCHAIN  1")
        )
     );
};
```

key()

Type:	template
Default:	empty string

Description: The key of the partition under which the message is published. You can use templates to change the topic dynamically based on the source or the content of the message, for example, `key("${PROGRAM}")`.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

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 by using the name, 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.

message()

Type:	message template
Default:	`$ISODATE $HOST $MSGHDR$MSG`

Description: The message as published to Apache Kafka. You can use templates and template functions (for example, `format-json()`) to format the message, for example, `message("$(format-json --scope rfc5424 --exclude DATE --key ISODATE)")`.

For details on formatting messages in JSON format, see format-json.

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.

persist-name()

Type:	string
Default:	

Description:If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error happens if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

poll-timeout()

Type:	integer in **`milliseconds`**
Default:	1000

Description: Specifies the frequency your syslog-ng PE queries the Kafka client about the amount of messages sent since the last `poll-timeout ()`. In case of multithreading, the first syslog-ng PE worker is responsible for `poll-timeout()`.

retries()

Type:	number (of attempts)
Default:	3

Description: If syslog-ng PE cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, syslog-ng PE will wait for `time-reopen()` time, then tries sending the message again.

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 Timezones and daylight saving.

The timezone can be specified by using the name, 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.

sync-send()

Type:	yes | no
Default:	no

Description: When `sync-send()` is set to `"yes"`, syslog-ng PE sends the message reliably: it sends a message to the Kafka server, then waits for a reply. 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 will wait for `time-reopen()` time, then tries sending the message again.

This method ensures reliable message transfer, but is very slow.

When `sync-send()` is set to `"no"`, syslog-ng PE sends messages asynchronously, and receives the response asynchronously. In case of a problem, syslog-ng PE cannot resend the messages.

NOTE: The underlying Kafka client (that is, librdkafka) may retry sending messages to syslog-ng PE independently several times.

This method is fast, but the transfer is not reliable. Several thousands of messages can be lost before syslog-ng PE recognizes the error.

CAUTION:

Hazard of data loss!

If `sync-send()` is set to `"no"`, the messages passed to the Kafka client can be lost.

To avoid data loss, One Identity recommends that you set `sync-send()` to `"yes"`, as this setting delivers messages to the Kafka client more reliably.

NOTE: If you want to use the `sync-send()` option set to `"yes"`, One Identity recommends that you use librdkafka version 1.4.0 or higher, and a Kafka server with version number 0.11.0 or higher.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Type:	number (seconds)
Default:	60

Description: Optional parameter.

If message sending fails, syslog-ng PE retries sending the messages for `retries()` time (`3` times by default) before waiting for `time-reopen()` time to try sending it again.

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 Date-related macros.

The timezone can be specified by using the name, 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.

topic()

Type:	string or template
Default:	N/A

Description: Required option. The Kafka topic under which the message is published. You can use templates to change the topic dynamically based on the source or the content of the message, for example, `topic("${HOST}")`.

NOTE: Valid topic names for the `topic()` and `fallback-topic()` options have the following limitations:

- The topic name must contain characters within the pattern `[-._a-zA-Z0-9]`.
- The length of the topic name must be between 1 and 249 characters.

NOTE: If you use templates with the `topic()` option, configuring the fallback-topic() option is also required.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

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.

workers()

Type:	integer
Default:	1

Description: The workers are only responsible for formatting the messages that need to be delivered to the Kafka clients. Configure this option only if your Kafka clients have many threads and they do not receive enough messages. If you set the `sync-send()` option to `yes`, the number of workers is automatically set to `1`.

NOTE: Kafka clients have their own threadpool, entirely independent from any syslog-ng PE settings. The `workers()` option has no effect on this threadpool.

`logstore`: 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 cipher() and 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 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 logstore() destination options.

CAUTION:

Hazard of data loss! If your log files are on an NFS-mounted network file system, see Using syslog-ng PE with NFS or CIFS (or SMB) file system for log files.

Declaration

```
logstore(filename options());
```

Example: 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("example")
              group("example")
              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 The logstore tool manual page.

TIP: The `lgstool` utility is available for Microsoft Windows operating systems at the Downloads page.

CAUTION:

For files that are in use by syslog-ng, the last chunk that is open cannot be read.

Journal files

The syslog-ng PE application 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 that 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 inaccessible, 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 logstore tool manual page.

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 approximately 2GB for the 500 hosts). To limit the memory use of journals, adjust the `logstore-journal-shmem-threshold()` global option (by default, it is 512MB).

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 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: 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 The syslog-ng manual page 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`.

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 when creating files or socket files.

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`.

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()`.

Starting with version 7.0.93.16, the default value of this option is -1, so syslog-ng PE does not change the ownership, unless explicitly configured to do so.

dir-perm()

Type:	number
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
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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
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: 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: In addition, consider the following:

- At least one journal block for every logstore file open is mapped into the memory. For details on logstore journals, see 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: 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
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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

mongodb: 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 mongodb() destination options.

NOTE: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see 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 Specifying data types in value-pairs.

Example: 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.

```
destination  d_mongodb  {
       mongodb(
              uri("mongodb://localhost:27017/syslog")
              collection("messages")
              value-pairs(
                     scope("selected-macros"  "nv-pairs"  "sdata")
              )
       );
 };
```

The following example shows the same setup using the deprecated libmongo-client syntax, 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")
              )
       );
 };
```

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 this 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: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see 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). The `collection()` option supports template functions and macros as well.

Example: using the mongodb() destination with a template embedded in the collection() option

Using the following example configuration, the `mongodb()` destination sends incoming messages into separate MongoDB collections (for example, `localhost_messages` and `anotherhost_messages`) based on the `HOST` field of the message :

```
mongodb(
    uri
("mongodb://host/syslog?wtimeoutMS=10000&socketTimeoutMS=10000&conne
ctTimeoutMS=10000&serverSelectionTimeoutMS=5000")
    collection("${HOST}_messages")
    workers(8)
 );
```

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.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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
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.

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.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

uri()

Type:	string
Default:	mongodb://127.0.0.1:27017/syslog?wtimeoutMS=60000&socketTimeoutMS=60000&connectTimeoutMS=60000

Description: Refer to the MongoDB URI format documentation for detailed syntax.

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 Structuring macros, metadata, and other value-pairs.

NOTE: Empty keys are not logged.

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 Specifying data types in value-pairs.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

network: 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.

- 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.
- 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 Encrypting log messages with TLS.

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: 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 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, seeBSD-syslog or legacy-syslog messages). The `network()` driver supports sending messages using the 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 file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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()

Description: Available only in syslog-ng Premium Edition version 7.0.93.17 and later. For details about how client-side failover works, see Client-side failover.

servers()

Type:	list of IP addresses and fully-qualified domain names
Default:	empty

Description: Specifies a secondary destination server where log messages are sent if the primary server becomes inaccessible. To list several failover servers, separate the address of the servers with comma. By default, syslog-ng PE waits for the a server before switching to the next failover server is set in the `time-reopen()` option.

If `failback()` is not set, syslog-ng PE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

failback()

Description: Available only in syslog-ng Premium Edition version 7.0.103.17 and later.

When syslog-ng PE starts up, it always connects to the primary server first. In the `failover()` option there is a possibility to customize the failover modes.

Depending on how you set the `failback()` option, syslog-ng PE behaves as follows:

- **round-robin mode**: If `failback()` is not set, syslog-ng PE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

Example: round-robin mode

In the following example syslog-ng PE handles the logservers in round-robin fashion if the primary logserver becomes inaccessible (therefore `failback()` option is not set).

```
destination d_network {
     network(
          "primary-server.com"
          port(601)
          failover( servers("failover-server1", "failover-server2")
)
```

```
);
};
```

- **failback mode**: If `failback()` is set, syslog-ng PE attempts to return to the primary server.

  After syslog-ng PE connects a secondary server during a failover, it sends a probe every `tcp-probe-interval()` seconds towards the primary server. If the primary logserver responds with a TCP ACK packet, the probe is successful. When the number of successful probes reaches the value set in the `successful-probes-required()` option, syslog-ng PE tries to connect the primary server using the last probe.

  NOTE: syslog-ng PE always waits for the result of the last probe before sending the next message. So if one connection attempt takes longer than the configured interval, that is, it waits for connection time out, you may experience longer intervals between actual probes.

Example: failback mode

In the following example syslog-ng PE attempts to return to the primary logserver, as set in the `failback()` option: it will check if the server is accessible every `tcp-probe-interval()` seconds, and reconnect to the primary logserver after three successful connection attempts.

```
destination d_network_2 {
     network(
          "primary-server.com"
          port(601)
          failover(
               servers("failover-server1", "failover-server2")
               failback(
                    successful-probes-required()
                    tcp-probe-interval()
               )
          )
);
};
```

Default value for `tcp-probe-interval()`: 60 seconds

Default value for `successful-probes-required()`: 3

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 inaccessible.

Example: 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

ip-protocol()

Type:	number
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 `ip-protocol(4)`.

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)`. You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source  s_network_tcp  {
       network(
             transport("tcp")
             ip("::")
             ip-protocol(6)
             port(601)
       );
 };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is 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
Default:	0

Description: The port number to bind to. Messages are sent from this port.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of 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.xsyslog-ng PE 3.3 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. Note that setting the global `mark-mode()` to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 and later.

port() or destport()

Type:	number
Default:	514

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make syslog-ng send messages to a broadcast address. 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
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
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. It is useful when you want to perform some kind of preprocessing via syslog-ng then forward messages to your central log management solution with the source address of the original sender. This option only works for UDP destinations though the original message can be received by TCP as well. This option is only available if syslog-ng was compiled using the `--enable-spoof-source` configuration option.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the `Last message repeated n times.` message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequential keepalive probes, regardless of the traffic exchanged in the connection. This option is equivalent to

/proc/sys/net/ipv4/tcp_keepalive_intvl. The default value is 0, which means using the kernel default.

CAUTION:

The tcp-keepalive-time(), tcp-keepalive-probes(), and tcp-keepalive-intvl() options only work on platforms which support the TCP_KEEPCNT, TCP_KEEPIDLE,and TCP_KEEPINTVL setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes() seconds.

Available in syslog-ng PE version 3.4 and later.

tcp-keepalive-probes()

Type:	number
Default:	0

Description: Specifies the number of unacknowledged probes to send before considering the connection dead. This option is equivalent to /proc/sys/net/ipv4/tcp_keepalive_probes. The default value is 0, which means using the kernel default.

CAUTION:

The tcp-keepalive-time(), tcp-keepalive-probes(), and tcp-keepalive-intvl() options only work on platforms which support the TCP_KEEPCNT, TCP_KEEPIDLE,and TCP_KEEPINTVL setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes() seconds.

Available in syslog-ng PE version 3.47.0 and later.

tcp-keepalive-time()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (in seconds) between the last data packet sent and the first keepalive probe. This option is equivalent to /proc/sys/net/ipv4/tcp_keepalive_time. The default value is 0, which means using the kernel default.

CAUTION:

The tcp-keepalive-time(), tcp-keepalive-probes(), and tcp-keepalive-intvl() options only work on platforms which support the TCP_KEEPCNT, TCP_KEEPIDLE,and TCP_KEEPINTVL setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes() seconds.

Available in syslog-ng PE version 3.4 and later.

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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 TLS options.

transport()

Type:	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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

`pipe`: 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 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: 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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
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.xsyslog-ng PE 3.3 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. Note that setting the global mark-mode() to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 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
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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

`program`: 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 program() destination options.

Declaration

```
program(command_to_run);
```

NOTE: When configuring the `program()` driver, consider the following:

- 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: 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} ${MESSAGE}\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 file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

inherit-environment()

Type:	yes|no
Default:	yes

Description: By default, when `program()` starts an external application or script, it inherits the entire environment of the parent process (that is, syslog-ng PE). Use `inherit-environment(no)` to prevent this.

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.xsyslog-ng PE 3.3 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. Note that setting the global `mark-mode()` to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 and later.

Note that in earlier versions of syslog-ng PE, the default for the mark-mode of the `program` destination was `none`. Now it defaults to the global setting, so the `program` destination will emit a MARK message every `mark-freq` interval. To avoid such messages, set the `mark-mode()` option of the destination to `none`.

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

python: writing custom Python destinations

The Python destination allows you to write your own destination in Python.

The following points apply to using Python blocks in syslog-ng PE in general:

- Only the default Python modules are available (that is, you cannot import external Python modules, and One Identity does not support using external Python modules).
- The syslog-ng PE application uses its own Python interpreter (shipped with the default syslog-ng PE installation) instead of the system's Python interpreter.
- The syslog-ng PE application is shipped with Python version 3.8.
- The Python block must be a top-level block in the syslog-ng PE configuration file.
- If you store the Python code in a separate Python file and only include it in the syslog-ng PE configuration file, make sure that the PYTHON_PATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start syslog-ng PE manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`

  In production, when syslog-ng PE starts on boot, you must configure your startup script to include the Python path. The exact method depends on your operating system. For recent Red Hat Enterprise Linux, Fedora, and CentOS distributions that use systemd, the `systemctl` command sources the `/etc/sysconfig/syslog-ng` file before starting syslog-ng PE. (On openSUSE and SLES, `/etc/sysconfig/syslog` file.) Append the following line to the end of this file: `PYTHONPATH="<path-to-your-python-file>"`, for example, `PYTHONPATH="/opt/syslog-ng/etc"`
- The Python object is initiated every time when syslog-ng PE is started or reloaded.

  CAUTION:

  If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.
- The Python block can contain multiple Python functions.
- Using Python code in syslog-ng PE can significantly decrease the performance of syslog-ng PE, especially if the Python code is slow. In general, the features of syslog-ng PE are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The syslog-ng PE application does not do any of this.

- Python error messages are available in the `internal()` source of syslog-ng PE.
- You can access the name-value pairs of syslog-ng PE directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of syslog-ng PE. For details, see Logging from your Python code.

- **Support disclaimer**

  CAUTION:

  This is a Preview Feature, which provides an insight to planned enhancements to functionality in the product. Consider this Preview Feature a work in progress, as it may not represent the final design and functionality.

  This feature has completed QA release testing, but its full impact on production systems has not been determined yet, and potential future changes in functionality and the user interface may result in compatibility issues in your current settings.

  One Identity recommends the following:

  - **Consider the potential risks when using this functionality in a production environment.**
  - **Consider the Support Policy on Product Preview Features before using this functionality in a production environment.**
  - **Closely and regularly keep track of official One Identity announcements about potential changes in functionality and the user interface. If these potential changes affect your configuration, check the changes you have to make in your configuration, otherwise your syslog-ng PE application may not start after upgrade.**
  - **Always perform tests prior to upgrades in order to avoid the risks mentioned.**

  However, you are welcome to try this feature and if you have any feedback, Contact One Identity.

  Support Policy on Product Preview Features

  The One Identity Support Team will:

  - **Accept and review each service request opened regarding a Preview Feature.**
  - **Consider all service requests relating to a Preview Features as severity level 3.**
  - **Provide best effort support to resolve any issues relating to a Preview Feature.**
  - **Work with customers to log any product defects or enhancements relating to Preview Features.**
  - **Not accept requests for escalations regarding Preview Features.**
  - **Not provide after-hours support for Preview Features.**

  Using Python in syslog-ng PE is recommended only if you are familiar with both Python and syslog-ng PE. One Identity is not responsible for the quality, resource requirements, or any bugs in the Python code, nor any syslog-ng PE crashes,

message losses, or any other damage caused by the improper use of this feature, unless explicitly stated in a contract with One Identity.

NOTE: Starting with 7.0.193.0.26, syslog-ng PE assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your syslog-ng PE configuration, add a unique `persist-name()` to each source or destination, otherwise syslog-ng PE will not start. For example:

```
log {
       source { python(class(PyNetworkSource) options("port" "8080")
persist-name("<unique-string>"); };
       source { python(class(PyNetworkSource) options("port" "8081")); };
    };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:

```
from syslogng import LogSource
  class PyNetworSource(LogSource):
    @staticmethod
    def generate_persist_name(options):
        return options["port"]
    def run(self):
        pass
    def request_exit(self):
        pass
```

Declaration

Python destinations consist of two parts. The first is a syslog-ng PE destination object that you define in your syslog-ng PE configuration and use in the log path. This object references a Python class, which is the second part of the Python destination. The Python class processes the log messages it receives, and can do virtually anything that you can code in Python. You can either embed the Python class into your syslog-ng PE configuration file, or store it in an external Python file.

```
destination <name_of_the_python_destination>{
    python(
        class("<name_of_the_python_class_executed_by_the_destination>")
    );
};

python {
class <name_of_the_python_class_executed_by_the_destination>(object):

    def open(self):
        """Open a connection to the target service
```

```
        Should return False if opening fails"""
        return True

    def close(self):
        """Close the connection to the target service"""
        pass

    def is_opened(self):
        """Check if the connection to the target is able to receive messages"""
        return True

    def init(self, options):
        """This method is called at initialization time

        Should return false if initialization fails"""
        return True

    def deinit(self):
        """This method is called at deinitialization time"""
        pass

    def send(self, msg):
        """Send a message to the target service

        It should return True to indicate success. False will suspend the
        destination for a period specified by the time-reopen() option.
        This will be repeated for the same message retries() times.

        Alternatively, it can return the following integer values:
        self.SUCCESS: message sending was successful (same as boolean True)
        self.ERROR: message sending was unsuccessful (same as boolean False)
        self.DROP: the message cannot be sent, it should be dropped immediately
        self.QUEUED: the message is not sent immediately, it will be sent in a
batch with the flush method
        self.NOT_CONNECTED: the message is put back into the queue, the open
method will be called until it succeeds
        self.RETRY: the message is put back into the queue, to retry send it
retries() times, then fallback to self.NOT_CONNECTED
        """
        return True
};
```

Methods of the python() destination

`init(self, options)` method (optional)

The syslog-ng PE application initializes Python objects every time when it is started or reloaded. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your source to work.

CAUTION:

If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.

When this method returns with False, syslog-ng PE does not start. It can be used to check options and return False when they prevent the successful start of the source.

`options`: This optional argument contains the contents of the `options()` parameter of the syslog-ng PE configuration object as a Python dictionary.

`is_opened(self)` method (optional)

Checks if the connection to the target is able to receive messages, and should return True if it is. For details, see Error handling in the python() destination.

`open(self)` method (optional)

The `open(self)` method opens the resources required for the destination, for example, it initiates a connection to the target service. It is called after `init()` when syslog-ng PE is started or reloaded. If `send()` returns with an error, syslog-ng PE calls `close()` and `open()` before trying to send again.

If `open()` fails, it should return the False value. In this case, syslog-ng PE retries it every `time-reopen()` seconds. By default, this is 1 second for Python sources and destinations, the value of `time-reopen()` is not inherited from the global option. For details, see Error handling in the python() destination.

`send(self, message)` method (mandatory)

The `send` method sends a message to the target service. It should return True to indicate success.

This is the only mandatory method of the destination.

If a message cannot be delivered after the number of times set in `retries()` (by default: 3), syslog-ng PE drops the message and continues with the next message. For details, see Error handling in the python() destination.

`close(self)` method (optional)

Close the connection to the target service. Usually it is called right before `deinit()` when stopping or reloading syslog-ng PE. It is also called when `send()` fails.

The `deinit(self)` method (optional)

This method is executed when syslog-ng PE is stopped or reloaded. This method does not return a value.

CAUTION:

If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.

Error handling in the python() destination

The Python destination handles errors as follows.

1. Currently syslog-ng PE ignores every error from the `open` method until the first log message arrives to the Python destination. If the fist message has arrived and there was an error in the `open` method, syslog-ng PE starts calling the `open` method every `time-reopen()` second, until opening the destination succeeds.
2. If the `open` method returns without error, syslog-ng PE calls the `send` method to send the first message.
3. If the `send` method returns with an error, syslog-ng PE calls the `is_opened` method.
    - If the `is_opened` method returns an error, syslog-ng PE starts calling the `open` method every `time-reopen()` second, until opening the destination succeeds.
    - Otherwise, syslog-ng PE calls the `send` method again.
4. If the `send` method has returned with an error `retries()` times and the `is_opened` method has not returned any errors, syslog-ng PE drops the message and attempts to process the next message.

Example: Write logs into a file

The purpose of this example is only to demonstrate the basics of the Python destination, if you really want to write log messages into text files, use the file destination instead.

The following sample code writes the body of log messages into the `/tmp/example.txt` file. Only the `send()` method is implemented, meaning that syslog-ng PE opens and closes the file for every message.

```
destination d_python_to_file {
       python(
              class("TextDestination")
       );
};
log {
       source(src);
       destination(d_python_to_file);
};
python {
class TextDestination(object):
       def send(self, msg):
              self.outfile = open("/tmp/example.txt", "a")
              self.outfile.write("MESSAGE = %s\n" % msg["MESSAGE"])
              self.outfile.flush()
              self.outfile.close();
              return True
};
```

The following code is similar to the previous example, but it opens and closes the file using the `open()` and `close()` methods.

```
destination d_python_to_file {
    python(
        class("TextDestination")
    );
};
log {
    source(src);
    destination(d_python_to_file);
};
python {
class TextDestination(object):
    def open(self):
        try:
            self.outfile = open("/tmp/example.txt", "a")
            return True
        except:
            return False

    def send(self, msg):
        self.outfile.write("MESSAGE = %s\n" % msg["MESSAGE"])
        self.outfile.flush()
        return True
```

```
    def close(self):
        try:
            self.outfile.flush()
            self.outfile.close();
            return True
        except:
            return False
};
```

For a more detailed example about sending log messages to an MQTT (Message Queuing Telemetry Transport) server, see the Writing Python destination in syslog-ng: how to send log messages to MQTT blog post.

For the list of available optional parameters, see python() destination options.

python() destination options

The Python destination allows you to write your own destination in Python. The `python()` destination has the following options. The `class()` option is mandatory. For details on writing destinations in Python, see python: writing custom Python destinations.

class()

Type:	string
Default:	N/A

Description: The name of the Python class that implements the destination, for example:

```
python(
       class("MyPythonDestination")
 );
```

If you want to store the Python code in an external Python file, the `class()` option must include the name of the Python file containing the class, without the path and the .py extension, for example:

```
python(
       class("MyPythonfilename.MyPythonDestination")
 );
```

For details, see Python code in external files

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the

timestamps after 6 digits.

log-fifo-size()

Type:	number
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.

options()

Type:	string
Default:	N/A

Description: This option allows you to pass custom values from the configuration file to the Python code. Enclose both the option names and their values in double-quotes. The Python code will receive these values during initialization as the `options` dictionary. For example,

you can use this to set the IP address of the server from the configuration file, so it is not hard-coded in the Python object.

```
python(
       class("MyPythonClass")
       options(
              "host"  "127.0.0.1"
              "port"  "1883"
              "otheroption"  "value")
 );
```

For example, you can refer to the value of the host field in the Python code as `options["host"]`. Note that the Python code receives the values as strings, so you might have to cast them to the type required, for example: `int(options["port"])`

persist-name()

Type:	string
Default:	None

Description: If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

NOTE: Starting with 7.0.193.0.26, syslog-ng PE assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your syslog-ng PE configuration, add a unique `persist-name()` to each source or destination, otherwise syslog-ng PE will not start. For example:

```
log  {
       source  {  python(class(PyNetworkSource)  options("port"  "8080")
persist-name("<unique-string>);  };
       source  {  python(class(PyNetworkSource)  options("port"  "8081"));  };
    };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:

```
from syslogng import LogSource
  class PyNetworSource(LogSource):
    @staticmethod
    def generate_persist_name(options):
        return options["port"]
    def run(self):
        pass
    def request_exit(self):
        pass
```

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

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 Structuring macros, metadata, and other value-pairs.

NOTE: Empty keys are not logged.

You can use this option to limit which name-value pairs are passed to the Python code for each message. Note that if you use the `value-pairs()` option, the Python code receives the specified value-pairs as a Python dict. Otherwise, it receives the message object. In the following example, only the text of the log message is passed to Python.

```
destination  d_python_to_file  {
       python(
             class("pythonexample.TextDestination")
             value-pairs(key(MESSAGE))
       );
};
```

sentinel(): Sending logs to the Microsoft Azure Sentinel cloud

Microsoft Azure Sentinel is Microsoft's native cloud based SIEM solution. Beside Microsoft's own cloud services, it can accept log messages from external sources. Microsoft provides 26 predefined Data connectors and a HTTP Data Collector API for further integrations. Using this public HTTP REST interface, syslog-ng Premium Edition (syslog-ng PE) can ingest log messages directly to Microsoft Azure Sentinel cloud using the `http()` destination driver.

For more information about Microsoft Azure Sentinel, see Microsoft Azure: Azure Sentinel documentation.

For more information about Data connectors used in Microsoft Azure Sentinel, see Microsoft Azure: Connect data sources.

CAUTION:

Hazard of data loss!

The `sentinel()` destination's `fields()` parameter is optional and has the following default values:

```
fields("Computer=$HOST HostName=$HOST ProcessID=$PID Syslo-
gMessage=$MSGHDR$MSG Facility=$FACILITY SeverityLevel=$LEVEL HostIP-
P=$SOURCEIP EventTime=$S_ISODATE")
```

Although it is possible to customize these default values, an incorrect configuration may result in log loss. Therefore, One Identity recommends that you do not customize the default values of the `fields()` parameter unless you know exactly what you are doing.

Limitations

The current implementation of the `sentinel()` destination has the following limitations:

- Only the PUT and the POST methods are supported.
- HTTPS connections, as well as password-based and certificate-based authentication, are supported.
- If the server returns a status code beginning with 4 (for example, 404) to the POST or PUT request, syslog-ng PE drops the message without attempting to resend it.

NOTE: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Declaration

```
destination d_sentinel {
       sentinel(
             workspace-id("<MS provided Workspace ID / UUID>")
             auth-secret("<MS provided Shared key / Secret>")
       );
 }
```

Configuring the sentinel() destination to send logs to the Microsoft Azure Sentinel cloud

Starting with version 7.0.19, syslog-ng Premium Edition (syslog-ng PE) can ingest log messages directly to Microsoft Azure Sentinel by using Microsoft Azure Sentinel's public HTTP Data Collector API interface.

The syslog-ng PE application can directly post log messages to the Microsoft Azure Sentinel cloud using the `http()` destination driver.

Prerequisites

To configure syslog-ng PE to forward messages to Microsoft Azure Sentinel, you must have an active Microsoft Azure Sentinel Workspace. For more information, see Microsoft Azure Sentinel: Quickstart: On-board Azure Sentinel.

Getting the required credentials to configure syslog-ng PE as a Data Connector for Microsoft Azure Sentinel

While configuring your Microsoft Azure Sentinel Workspace according to the Microsoft Azure Sentinel: Quickstart: On-board Azure Sentinel guide's Connect data sources section, you will find certain credentials of the selected Microsoft Azure Sentinel Workspace (namely, **WORKSPACE ID** and **PRIMARY KEY**) that you should make a note of. The **WORKSPACE ID** and **PRIMARY KEY** credentials are required so that syslog-ng PE can authenticate to Microsoft servers.

To gather the required credentials to configure syslog-ng PE as a Data connector for Microsoft Azure Sentinel,

1. Log in to your Microsoft Azure Sentinel account, then navigate to **Dashboard > Azure Sentinel workspaces**.

2. Select the workspace you want to use, then under the selected workspace, select **Data Connectors**.
3. Select **Syslog** from the list of connectors.

   A new panel will open on the right.
4. Select **Open connector page** from the new panel on the right.
5. Navigate back to **Dashboard > Azure Sentinel workspaces > Azure Sentinel - Data connectors > Syslog**.
6. Under **Instructions**, select **Open your workspace advanced settings configuration**.
7. Navigate to **Dashboard > Azure Sentinel - Data connectors > Syslog > Advanced settings.**
8. Copy the following information from **Connected Sources > Linux Servers**:
   - **WORKSPACE ID**

     In the syslog-ng PE configuration, this will function as the `workspace-id` parameter.
   - **PRIMARY KEY**

     In the syslog-ng PE configuration, this will function as the `auth-secret` parameter.

Example: Using Azure Sentinel credentials in the `sentinel()` destination

In your syslog-ng PE configuration, you can use the credentials you have gathered like this:

```
d_sentinel {
       sentinel(
             workspace-id("01234567-89ab-cdef-0123-456789abcdef") //
Provided by Microsoft: unique hexadecimal number, identifying your
Sentinel instance.
             auth-secret
("MDEyMzQ1Njc4OWFiY2RlZjAxMjM0NTY3ODlhYmNkZWYwMTIzNDU2Nzg5YWJjZGVmMDEyMzQ1
Njc4OWFiY2RlZgo=") // Provided by Microsoft: Base64-encoded secret,
identifying your application.

             # optional
             log-type("Syslog_CL")
       );
};
```

CAUTION:

Hazard of data loss!

The `sentinel()` destination's `fields()` parameter is optional and has the following default values:

```
fields("Computer=$HOST HostName=$HOST ProcessID=$PID Syslo-
gMessage=$MSGHDR$MSG Facility=$FACILITY SeverityLevel=$LEVEL HostIP-
P=$SOURCEIP EventTime=$S_ISODATE")
```

Although it is possible to customize these default values, an incorrect configuration may result in log loss. Therefore, One Identity recommends that you do not customize the default values of the `fields()` parameter unless you know exactly what you are doing.

Log types

On the Microsoft Azure Sentinel user interface, you can filter log events by referring to `log-type` parameters. Although the `log-type` parameters query information with their custom query language (namely, Kusto), they are essentially table fields in a database.

For more information about `log-type` parameters, see Microsoft Azure: Write data to Log Analytics repository.

For more information about the Kusto query language, see Microsoft Azure: Query Language: Overview.

The `log-type` parameter is mandatory for the Microsoft Azure Sentinel HTTP Data Collector API interface, and for syslog-ng Premium Edition (syslog-ng PE), its default value is `Syslog_CL`.

NOTE: While Microsoft Azure Sentinel accepts log events from external sources, it also has built-in log types. To differentiate external and built-in log types, Azure Sentinel automatically applies the Custom Log-type (`_CL`) suffix to log events that originate from external sources. Although the `log-type` parameter is optional (unlike `workspace-id` and `auth-secret`, which are mandatory), keep in mind that Microsoft Azure Sentinel will automatically attach the `_CL` suffix to all log events originating from external sources. For example, log events that originate from syslog-ng PE will be named `Syslog_CL` by default.

NOTE: If you want to store messages in different log types that all connect to the same `workspace-id`, you can create multiple `sentinel()` destinations that have a common `workspace-id`, but different `log-type` parameters.

Example syslog-ng PE message

When forwarding messages to Microsoft Azure Sentinel, syslog-ng PE automatically populates the following fields:

```
{
       "SyslogMessage": "<34>Oct 11 22:14:15 sentineluser prog1:
message_begin 2020-02-10 08:29:02:329951-default message_end\n",
       "SeverityLevel": "notice",
       "ProcessID": "",
       "HostName": "myhost",
       "HostIP": "127.0.0.1",
       "Facility": "user",
       "EventTime": "2020-02-10T09:29:48+01:00",
       "Computer": "ubuntu"
 }
```

sentinel() destination options

The `sentinel()` destination of syslog-ng Premium Edition (syslog-ng PE) can directly post log messages to Microsoft Azure Sentinel using its HTTP Data Collector API. The `sentinel()` destination has the following options.

auth-secret()

Type:	Provided by Microsoft: Base64-encoded secret, identifying your application. On the syslog-ng PE side, it is handled as string.
Default:	N/A

Description: A unique option for syslog-ng PE's `sentinel()` destination. This parameter is required so that syslog-ng PE can authenticate to Microsoft servers. You can collect the required credentials on your Microsoft Azure Sentinel Workspace.

For more information, see Getting the required credentials to configure syslog-ng PE as a Data Connector for Microsoft Azure Sentinel.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, syslog-ng PE sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in syslog-ng PE version 7.0.12 and later.

batch-lines()

Type:	number [lines]
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

body()

Type:	string or template
Default:	

Description: The body of the HTTP request, for example, `body("${ISODATE} ${MESSAGE}")`. You can use strings, macros, and template functions in the body. If not set, it will contain the message received from the source by default.

body-prefix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts at the beginning of the body of the HTTP request, before the log message. Available in syslog-ng PE version 7.0.11 and later.

body-suffix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts to the end of the body of the HTTP request, after the log message. Available in syslog-ng PE version 7.0.11 and later.

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. For an example, see Configuring TLS on the syslog-ng clients. The syslog-ng PE application uses the CA certificates in this directory to validate the certificate of the peer.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
       http(
               url("http://127.0.0.1:8080")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
                       peer-verify(yes|no)
                       ssl-version(<the permitted SSL/TLS version>)
               )
       );
};
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the HTTPS server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
       http(
               url("http://127.0.0.1:8080")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
```

```
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 set in the `key-file()` option. The syslog-ng PE application uses this certificate to authenticate the syslog-ng PE client on the destination server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that syslog-ng PE uses

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")
```

For a list of available algorithms, execute the `openssl ciphers -v` command. The first column of the output contains the name of the algorithms to use in the `cipher-suite()` option, the second column specifies which encryption protocol uses the algorithm (for example, `TLSv1.2`). That way, the `cipher-suite()` also determines the encryption protocol used in the connection: to disable SSLv3, use an algorithm that is available only in TLSv1.2, and that both the client and the server supports. You can also specify the encryption protocols using ssl-options().

You can also use the following command to automatically list only ciphers permitted in a specific encryption protocol, for example, `TLSv1.2`:

```
echo "cipher-suite(\"$(openssl ciphers -v | grep TLSv1.2 | awk '{print $1}' |
xargs echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
       http(
               url("http://127.0.0.1:8080")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
```

```
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, syslog-ng PE separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option. Available in syslog-ng PE version 3.187.0.11 and later.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000
Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.	

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
        )
    );
};
```

headers()

Type:	string list
Default:	

Description: Custom HTTP headers to include in the request, for example, `headers ("HEADER1: header1", "HEADER2: header2")`. If not set, only the default headers are included, but no custom headers.

The following headers are included by default:

- X-Syslog-Host: <host>
- X-Syslog-Program: <program>
- X-Syslog-Facility: <facility>
- X-Syslog-Level: <loglevel/priority>

hook-commands()

Description: This option makes it possible to execute external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to execute external applications.

Using the hook-commands() when syslog-ng PE starts or stops

To execute an external program when syslog-ng PE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A

Description: Defines the external program that is executed when syslog-ng PE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed when syslog-ng PE stops.

Using the hook-commands() when syslog-ng PE reloads

To execute an external program when the syslog-ng PE configuration is initiated or torn down (for example, on startup/shutdown or during a syslog-ng PE reload), use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.

Example: Using the `hook-commands()` with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically when syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow (otherwise the port is closed).

```
source  {
    network(transport(udp)
       hook-commands(
          startup("iptables  -I  LOGCHAIN  1  -p  udp  --dport  514  -j
ACCEPT")
          shutdown("iptables  -D  LOGCHAIN  1")
       ;)
    );
 };
```

key-file()

Accepted values:	Filename
Default:	none

Description: Path and name of a file that contains a private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng PE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng PE client on the destination server.

The `sentinel()` destination supports only unencrypted key files (that is, the private key cannot be password-protected).

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
       http(
               url("http://127.0.0.1:8080")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
```

```
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

log-type()

Type:	Microsoft Azure Sentinel parameter and an optional syslog-ng PE configuration option
Default:	`Syslog_CL` for syslog-ng PE

Description:

On the Microsoft Azure Sentinel user interface, you can filter log events by referring to `log-type` parameters. Although the `log-type` parameters query information with their custom query language (namely, Kusto), they are essentially table fields in a database.

The `log-type` parameter is mandatory for the Microsoft Azure Sentinel HTTP Data Collector API interface, and for syslog-ng Premium Edition, its default value is `Syslog_CL`.

> NOTE: While Microsoft Azure Sentinel accepts log events from external sources, it also has built-in log types. To differentiate external and built-in log types, Azure Sentinel automatically applies the Custom Log-type (`_CL`) suffix to log events that originate from external sources. Although the `log-type` parameter is optional (unlike `workspace-id` and `auth-secret`, which are mandatory), keep in mind that Microsoft Azure Sentinel will automatically attach the `_CL` suffix to all log events originating from external sources. For example, log events that originate from syslog-ng PE will be named `Syslog_CL` by default.

For more information, see Getting the required credentials to configure syslog-ng PE as a Data Connector for Microsoft Azure Sentinel and Log types.

method()

Type:	POST | PUT
Default:	POST

Description: Specifies the HTTP method to use when sending the message to the server.

password()

Type:	string
Default:	

Description: The password that syslog-ng PE uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	yes | no
Default:	yes

Description: Verification method of the peer. 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	*no (optional-untrusted)*	TLS-encryption	TLS-encryption	TLS-encryption
	yes (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.

persist-name()

Type:	string
Default:	None

*Description:*If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

The `proxy()` option enables you to configure the `sentinel` driver to use a specific HTTP proxy for all HTTP-based destinations, instead of using the proxy that is configured for the system.

If you do not set the `proxy()` option, the `sentinel` driver uses the `http_proxy` and `https_proxy` environment variables, as shown in CURLOPT_PROXY explained.

NOTE: Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.

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.

To handle HTTP error responses, if the HTTP server returns 5xx codes, syslog-ng PE will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, syslog-ng PE will drop the messages.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_http {
        http(
                url("http://127.0.0.1:8080")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to

avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

timeout()

Type:	number [seconds]
Default:	0

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded. By default, the timeout value is `0`, meaning that there is no timeout. Available in version 7.0.4 and later.

url()

Type:	URL or list of URLs
Default:	http://localhost/

Description: Specifies the hostname or IP address, and optionally the port number of the web service that can receive log data through HTTP. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://127.0.0.1:8000`

In case the server on the specified URL returns a redirect request, syslog-ng PE automatically follows maximum 3 redirects. Only HTTP-based and HTTPS-based redirections are supported.

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

user-agent()

Type:	string
Default:	syslog-ng [version]/libcurl[version]

Description: The value of the USER-AGENT header in the messages sent to the server.

user()

Type:	string
Default:	

Description: The username that syslog-ng PE uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	yes | no
Default:	no

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

workspace-id()

Type:	Provided by Microsoft: unique hexadecimal number, identifying your Sentinel instance. On the syslog-ng PE side, it is handled as string.

Default: N/A

Description: A unique option for syslog-ng PE's `sentinel()` destination. This parameter is required so that syslog-ng PE can authenticate to Microsoft servers. You can collect the required credentials on your Microsoft Azure Sentinel Workspace.

For more information, see Getting the required credentials to configure syslog-ng PE as a Data Connector for Microsoft Azure Sentinel.

snmp: 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.

The `snmp()` driver is available in syslog-ng PE version 7.0.203.22 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()` and `version()` options when using the SNMPv3 protocol. For the list of available optional parameters, see snmp() destination options.

NOTE:To use the `snmp()` driver, the `scl.conf` file must be included in your syslog-ng PE configuration:

```
@include "scl.conf"
```

Declaration

```
destination d_snmp {snmp(host() trap-obj() snmp-obj() ...);};
```

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: Using the snmp() destination driver

The following example defines an SNMP destination that uses the SNMPv2c protocol.

```
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.

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

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:	empty string

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 between 5 and 32 characters long, starting with `0x`. for example, `0xABABABABAB`.

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
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:	n/a

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: 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 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:	n/a

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 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.

`smtp`: Generating SMTP messages (email) from logs

The destination is aimed at a fully controlled local, or near-local, trusted SMTP server. The goal is to send mail to trusted recipients, through a controlled channel. It hands mails over to an SMTP server, and that is all it does, therefore the resulting solution is as reliable as sending an email in general. For example, syslog-ng PE does not verify whether the recipient exists.

The `smtp()` driver sends email messages triggered by log messages. The `smtp()` driver uses SMTP, without needing external applications. You can customize the main fields of the email, add extra headers, send the email to multiple recipients, and so on.

The `subject()`, `body()`, and `header()` fields may include macros which get expanded in the email. For more information on available macros see Macros of syslog-ng PE.

The `smtp()` driver has the following required parameters: `host()`, `port()`, `from()`, `to()`, `subject()`, and `body()`. For the list of available optional parameters, see smtp() destination options.

NOTE: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

NOTE: The `smtp()` destination driver is available only in syslog-ng PE 3.45 F2 and later.

Declaration

```
smtp(host() port() from() to() subject() body() options());
```

Example: Using the smtp() driver

The following example defines an `smtp()` destination using only the required parameters.

```
destination d_smtp {
       smtp(
             host("localhost")
             port(25)
             from("syslog-ng alert service" "noreply@example.com")
             to("Admin #1" "admin1@example.com")
             subject("[ALERT] Important log message of $LEVEL condition
received from $HOST/$PROGRAM!")
             body("Hi!\nThe syslog-ng alerting service detected the
following important log message:\n $MSG\n-- \nsyslog-ng\n")
       );
 };
```

The following example sets some optional parameters as well.

```
destination d_smtp {
    smtp(
        host("localhost")
        port(25)
        from("syslog-ng alert service" "noreply@example.com")
        to("Admin #1" "admin1@example.com")
```

```
        to("Admin #2" "admin2@example.com")
        cc("Admin BOSS" "admin.boss@example.com")
        bcc("Blind CC" "blindcc@example.com")
        subject("[ALERT] Important log message of $LEVEL condition
received from $HOST/$PROGRAM!")
        body("Hi!\nThe syslog-ng alerting service detected the following
important log message:\n $MSG\n-- \nsyslog-ng\n")
        header("X-Program", "$PROGRAM")
    );
};
```

Example: Simple email alerting with the `smtp()` driver

The following example sends an email alert if the eth0 network interface of the host is down.

```
filter f_linkdown {
      match("eth0: link down" value("MESSAGE"));
};
destination d_alert {
      smtp(
            host("localhost") port(25)
            from("syslog-ng alert service" "syslog@localhost")
            reply-to("Admins" "root@localhost")
            to("Ennekem" "me@localhost")
            subject("[SYSLOG ALERT]: eth0 link down")
            body("Syslog received an alert:\n$MSG")
      );
};

log {
      source(s_local);
      filter(f_linkdown);
      destination(d_alert);
};
```

smtp() destination options

The `smtp()` sends email messages using SMTP, without needing external applications. The `smtp()` destination has the following options:

body()

Type:	string
Default:	n/a

Description: The BODY field of the email. You can also use macros in the string. Use `\n` to start a new line. For example:

```
body("syslog-ng PE received the following alert from $HOST:\n$MSG")
```

bcc()

Type:	string
Default:	n/a

Description: The BCC recipient of the email (contents of the BCC field). You can specify the email address, or the name and the email address. Set the `bcc()` option multiple times to send the email to multiple recipients. For example: `bcc("admin@example.com")` or `bcc ("Admin" "admin@example.com")` or `bcc("Admin" "admin@example.com") bcc("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

cc()

Type:	string
Default:	n/a

Description: The CC recipient of the email (contents of the CC field). You can specify the email address, or the name and the email address. Set the `cc()` option multiple times to send the email to multiple recipients. For example: `cc("admin@example.com")` or `cc("Admin" "admin@example.com")` or `cc("Admin" "admin@example.com") cc("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
              )
       );
 };
```

from()

Type:	string
Default:	n/a

Description: The sender of the email (contents of the FROM field). You can specify the email address, or the name and the email address. For example:

```
from("admin@example.com")
```

or

```
from("Admin" "admin@example.com")
```

If you specify the `from()` option multiple times, the last value will be used. Instead of the `from()` option, you can also use `sender()`, which is just an alias of the `from()` option.

You can also use macros to set the value of this parameter.

header()

Type:	string
Default:	n/a

Description: Adds an extra header to the email with the specified name and content. The first parameter sets the name of the header, the second one its value. The value of the header can contain macros. Set the `header()` option multiple times to add multiple headers. For example:

```
header("X-Program", "$PROGRAM")
```

When using the header option, note the following points:

- Do not use the `header()` option to set the values of headers that have dedicated options. Use it only to add extra headers.
- If you set the same custom header multiple times, only the first will be added to the email, other occurrences will be ignored.
- It is not possible to set the DATE, Return-Path, Original-Recipient, Content-*, MIME-*, Resent-*, Received headers.

host()

Type:	hostname or IP address
Default:	n/a

Description: Hostname or IP address of the SMTP server.

NOTE: If you specify `host="localhost"`, syslog-ng PE will use a socket to connect to the local SMTP server. Use `host="127.0.0.1"` to force TCP communication between syslog-ng PE and the local SMTP server.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

port()

Type:	number
Default:	25

Description: The port number of the SMTP server.

reply-to()

Type:	string
Default:	n/a

Description: Replies of the recipient will be sent to this address (contents of the REPLY-TO field). You can specify the email address, or the name and the email address. Set the `reply-to()` option multiple times to send the email to multiple recipients. For example: `reply-to("admin@example.com")` or `reply-to("Admin" "admin@example.com")` or `reply-to ("Admin" "admin@example.com") reply-to("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

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.

subject()

Type:	string
Default:	n/a

Description: The SUBJECT field of the email. You can also use macros. For example:

```
subject("[SYSLOG ALERT]: Critical error message received from $HOST")
```

If you specify the `subject()` option multiple times, the last value will be used.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to

avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

to()

Type:	string
Default:	localhost

Description: The recipient of the email (contents of the TO field). You can specify the email address, or the name and the email address. Set the `to()` option multiple times to send the email to multiple recipients. For example: `to("admin@example.com")` or `to("Admin" "admin@example.com")` or `to("Admin" "admin@example.com") to("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

splunk-hec: Sending messages to Splunk HTTP Event Collector

Version 7.0.12 of syslog-ng PE can directly post log messages to a Splunk deployment using the HTTP Event Collector (HEC) over the HTTP and Secure HTTP (HTTPS) protocols.

HTTPS connection, as well as password- and certificate-based authentication is supported. The content of the events is sent in JSON format.

NOTE: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Declaration

```
d_splunk_hec  {
       splunk_hec(
             #mandatory
             index("<splunk-index-to-store-messages>")
             token("<event-collector-tokens>")
             url("http://<your-splunk-server>:8088/services/collector/event")
       );
 };
```

Prerequisites

- On your Splunk deployment, you must enable HTTP Event Collector (HEC).
- On your Splunk deployment, you must create a token for syslog-ng PE. You must use

this token in the `token()` option of your `splunk-hec()` destination. We recommend to use the `syslog` source type for the token.

For details, see Set up and use HTTP Event Collector in Splunk Web.

Example: Sending log data to Splunk

The following example defines a `splunk-hec()` destination.

```
d_splunk_hec  {
       splunk_hec(
              #  mandatory
              index("<splunk-index-to-store-messages>")
              token("<event-collector-tokens>")
              url("http://<your-splunk-
server>:8088/services/collector/event")

              #  optional
              batch_lines(25)
              workers(4)
              source("syslog-ng")
              sourcetype("${.app.name:-syslog}")
              delimiter("\n")
              time("$S_UNIXTIME.$S_MSEC")
              host("$HOST")
              event("$S_ISODATE  $HOST  $MSGHDR$MSG\n")
              timeout(10));
       );
};

log  {
       source(s_file);
       destination(d_splunk_hec);
       flags(flow-control);
};
```

Batch mode and load balancing

The `splunk-hec()` destination automatically sends multiple log messages in a single HTTP request, increasing the rate of messages that your Splunk deployment can consume. For details on adjusting and fine-tuning the batch mode of the `splunk-hec()` destination, see the following section.

Batch size

The `batch-lines()`, `batch-lines()`, and `batch-timeout()` options of the destination determine how many log messages syslog-ng PE sends in a batch. The `batch-lines()` option determines the maximum number of messages syslog-ng PE puts in a batch in. This can be limited based on size and time:

- syslog-ng PE sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. This ensures that the destination receives every message in a timely manner even if suddenly there are no more messages.
- syslog-ng PE sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Example: HTTP batch mode

In the following example, a batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
destination  d_splunk-hec  {
       splunk-hec(url("http://your-splunk-
server:8088/services/collector/event")
             index("<splunk-index-to-store-messages>")
             token("<event-collector-tokens>")
             url("http://your-splunk-
server:8088/services/collector/event")
             batch-lines(100)
             batch-bytes(512Kb)
             batch-timeout(10000)
       );
 };
```

Load balancing between multiple Splunk indexers

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages.

If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

Example: HTTP load balancing

The following destination sends log messages to 3 different Splunk indexer nodes. Each node is assigned a separate worker thread. A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
destination  d_splunk-hec  {
       splunk-hec(url("http://your-splunk-
server1:8088/services/collector/event"  "http://your-splunk-
server2:8088/services/collector/event"  "http://your-splunk-
server3:8088/services/collector/event")
              batch-lines(100)
              batch-bytes(512Kb)
              batch-timeout(20000)
              persist-name("d_splunk-hec-load-balance")
       );
 };
```

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

splunk-hec destination options

The `splunk-hec` destination of syslog-ng PE can directly post log messages to a Splunk deployment using the HTTP Event Collector (HEC) over the HTTP and Secure HTTP (HTTPS) protocols. The `splunk-hec` destination has the following options. The required options are: `index()`, `token()`, and `url()`.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, syslog-ng PE sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in syslog-ng PE version 3.197.0.12 and later.

For details on how this option influences batch mode, see splunk-hec: Sending messages to Splunk HTTP Event Collector on page 562

batch-lines()

Type:	number
Default:	25

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

For details on how this option influences batch mode, see splunk-hec: Sending messages to Splunk HTTP Event Collector on page 562

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer

starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

For details on how this option influences batch mode, see splunk-hec: Sending messages to Splunk HTTP Event Collector on page 562

body-prefix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts at the beginning of the body of the HTTP request, before the log message. Available in syslog-ng PE version 3.187.0.11 and later.

For details on how this option influences batch mode, see splunk-hec: Sending messages to Splunk HTTP Event Collector on page 562

body-suffix()

Accepted values:	string
Default:	none

Description: The string syslog-ng PE puts to the end of the body of the HTTP request, after the log message. Available in syslog-ng PE version 3.187.0.11 and later.

For details on how this option influences batch mode, see splunk-hec: Sending messages to Splunk HTTP Event Collector on page 562

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. For an example, see Configuring TLS on the syslog-ng clients. The syslog-ng PE application uses the CA certificates in this directory to validate the certificate of the peer.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
        splunk-hec(
                url("http://your-splunk-server3:8088/services/collector/event")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The syslog-ng PE application uses this certificate to validate the certificate of the HTTPS server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
        splunk-hec(
                url("http://your-splunk-server3:8088/services/collector/event")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
```

```
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 set in the `key-file()` option. The syslog-ng PE application uses this certificate to authenticate the syslog-ng PE client on the destination server. 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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
        splunk-hec(
                url("http://your-splunk-server3:8088/services/collector/event")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that syslog-ng PE uses

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")
```

For a list of available algorithms, execute the `openssl ciphers -v` command. The first column of the output contains the name of the algorithms to use in the `cipher-suite()` option, the second column specifies which encryption protocol uses the algorithm (for example, `TLSv1.2`). That way, the `cipher-suite()` also determines the encryption protocol used in the connection: to disable SSLv3, use an algorithm that is available only in TLSv1.2, and that both the client and the server supports. You can also specify the encryption protocols using ssl-options().

You can also use the following command to automatically list only ciphers permitted in a specific encryption protocol, for example, `TLSv1.2`:

```
echo "cipher-suite(\"$(openssl ciphers -v | grep TLSv1.2 | awk '{print $1}' |
xargs echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
       splunk-hec(
               url("http://your-splunk-server3:8088/services/collector/event")
               tls(
                       ca-dir("dir")
                       ca-file("ca")
                       cert-file("cert")
                       cipher-suite("cipher")
                       key-file("key")
```

```
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, syslog-ng PE separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option.

For details on how this option influences batch mode, see splunk-hec: Sending messages to Splunk HTTP Event Collector on page 562

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000
Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.	

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
              )
       );
 };
```

event()

Type:	template
Default:	${S_ISODATE} ${HOST} ${MSGHDR}${MSG}\n

Description: The body of the message that syslog-ng PE sends to Splunk. The content of the events (the value of the `${MSG}` macro) is sent in JSON format.

hook-commands()

Description: This option makes it possible to execute external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to execute external applications.

Using the hook-commands() when syslog-ng PE starts or stops

To execute an external program when syslog-ng PE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed when syslog-ng PE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed when syslog-ng PE stops.

Using the hook-commands() when syslog-ng PE reloads

To execute an external program when the syslog-ng PE configuration is initiated or torn down (for example, on startup/shutdown or during a syslog-ng PE reload), use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.

Example: Using the `hook-commands()` with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically when syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow (otherwise the port is closed).

```
source {
    network(transport(udp)
       hook-commands(
          startup("iptables -I LOGCHAIN 1 -p udp --dport 514 -j
ACCEPT")
          shutdown("iptables -D LOGCHAIN 1")
       ;)
    );
 };
```

host()

Type:	template
Default:	${HOST}

Description: The hostname for the message that syslog-ng PE sends to Splunk. In Splunk, the message will appear as it has been sent by this host.

index()

Accepted values:	string
Default:	None

Description: The name of the Splunk index where Splunk will store the messages received from syslog-ng PE. This option is mandatory for this destination.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

key-file()

Accepted values:	Filename
Default:	none

Description: Path and name of a file that contains a private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng PE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng PE client on the destination server.

This destination supports only unencrypted key files (that is, the private key cannot be password-protected).

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
        splunk-hec(
                url("http://your-splunk-server3:8088/services/collector/event")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

method()

Type:	POST | PUT
Default:	POST

Description: Specifies the HTTP method to use when sending the message to the server.

password()

Type:	string
Default:	

Description: The password that syslog-ng PE uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	yes | no
Default:	yes

Description: Verification method of the peer. 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	*no (optional-untrusted)*	TLS-encryption	TLS-encryption	TLS-encryption
	yes (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.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
       splunk-hec(
               url("http://your-splunk-server3:8088/services/collector/event")
               tls(
                       ca-dir("dir")
```

```
                    ca-file("ca")
                    cert-file("cert")
                    cipher-suite("cipher")
                    key-file("key")
                    peer-verify(yes|no)
                    ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

persist-name()

Type:	string
Default:	None

Description: If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

The `proxy()` option enables you to configure the `splunk-hec` driver to use a specific HTTP proxy for all HTTP-based destinations, instead of using the proxy that is configured for the system.

If you do not set the `proxy()` option, the `splunk-hec` driver uses the `http_proxy` and `https_proxy` environment variables, as shown in CURLOPT_PROXY explained.

> NOTE: Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.

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.

To handle HTTP error responses, if the HTTP server returns 5xx codes, syslog-ng PE will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, syslog-ng PE will drop the messages.

source()

Type:	string
Default:	syslog-ng

Description: The value of the source field in Splunk.

sourcetype()

Type:	string
Default:	.app.name:-syslog

Description: The source type that you set in Splunk for the token that syslog-ng PE uses. For details about source types, see the Splunk documentation.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

An alternative way to specify this option is to put it into a `tls()` block, together with any other TLS options. This allows you to separate these options and ensure better readability.

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. If you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration

```
destination d_splunk-hec {
        splunk-hec(
                url("http://your-splunk-server3:8088/services/collector/event")
                tls(
                        ca-dir("dir")
                        ca-file("ca")
                        cert-file("cert")
                        cipher-suite("cipher")
                        key-file("key")
                        peer-verify(yes|no)
                        ssl-version(<the permitted SSL/TLS version>)
                )
        );
};
```

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time()

Type:	template
Default:	${S_UNIXTIME}.${S_MSEC}

Description: The timestamp for the message that syslog-ng PE sends to Splunk.

timeout()

Type:	number [seconds]
Default:	10

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded.

token()

Accepted values:	string
Default:	None

Description: The Splunk HTTP Event Collector token that permits syslog-ng PE to send messages to Splunk. For details, see Set up and use HTTP Event Collector in Splunk Web. This option is mandatory for this destination.

url()

Type:	URL or list of URLs, for example, url("site1" "site2")
Default:	http://localhost:8088/services/collector/event

Description: Specifies the hostname or IP address and optionally the port number of the Splunk indexer. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://your-splunk-indexer.server:8088/services/collector/event`

This option is mandatory for this destination.

Make sure that the URL ends with `event`, this is the Splunk API endpoint that properly parses the messages sent by syslog-ng PE.

In case the server on the specified URL returns a redirect request, syslog-ng PE automatically follows maximum 3 redirects. Only HTTP and HTTPS based redirections are supported.

Starting with version 3.197.0.12, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, syslog-ng PE sends log messages to the specified URLs in a load-balance fashion. This means that syslog-ng PE sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order syslog-ng PE has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

CAUTION:

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

user-agent()

Type:	string
Default:	syslog-ng [version]/libcurl[version]

Description: The value of the USER-AGENT header in the messages sent to the server.

user()

Type:	string
Default:	

Description: The username that syslog-ng PE uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	yes | no
Default:	no

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation.

workers()

Type:	integer
Default:	4

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

sql(): Storing messages in an SQL database

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

NOTE: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Declaration

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

The `sql()` driver has the following required parameters: `type()`, `database()`, `table()`, `columns()`, and `values()`.

CAUTION:

The syslog-ng Premium Edition (syslog-ng PE) application requires read and write access to the SQL table, otherwise it cannot verify that the destination table exists.

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

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

The `sql()` driver is currently not available for every platform that is supported by syslog-ng PE. .

The `table` and `value` parameters can include macros to create tables and columns dynamically (for details, see Macros of syslog-ng PE).

CAUTION:

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

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

Example: Using the sql() driver

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

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

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

```
destination d_sql {
    sql(
        type(pgsql)
        host("logserver")
        username("syslog-ng")
        password("password")
        database("logs")
        table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
        columns("datetime varchar(16)", "host varchar(32)", "program
```

```
varchar(20)", "pid varchar(8)", "message  varchar(200)")
        values("${R_DATE}", "${HOST}", "${PROGRAM}", "${PID}",
"${MSGONLY}")
        indexes("datetime", "host", "program", "pid", "message")
    );
};
```

Using the sql() driver with an Oracle database

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

- There are two ways to set the hostname of the database server:
  - Set the hostname in the `tnsnames.ora` file, not in the `host` parameter of the `sql()` destination.

    If the `tnsnames.ora` file is not located in the /etc directory (or in the /var/opt/oracle directory on Solaris), set the following Oracle-related environment variables, so syslog-ng PE will find the file: `ORACLE_BASE`, `ORACLE_HOME`, and `ORACLE_SID`. For details, see the documentation of the Oracle Instant Client.

    Note that in this case you cannot use the same `database()` settings in more than one destination, because the `database()` option of the SQL driver is just a reference to the connection string of the `tnsnames.ora` file. To overcome this problem, you can duplicate the connections in the `tnsnames.ora` file under a different name, and use a different table in each Oracle destination in syslog-ng PE.
  - Set the hostname in the `host()` option of the destination, and set the `ignore-tns-ora()` to yes. In this case, syslog-ng PE ignores the `tnsnames.ora` file, and you can use the same database settings in multiple destinations. Note that the `ignore-tns-ora()` option is available in syslog-ng Premium Edition version 7.0.93.16.
- As certain database versions limit the maximum length of table names, macros in the table names should be used with care.
- In the current version of syslog-ng PE, the types of database columns must be explicitly set for the Oracle destination. The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng PE, therefore it is usually recommended to use the `varchar2` or `clob` column type. (The maximum length of the messages can be set using the `log-msg-size()` option.) For details, see the following example.
- The Oracle Instant Client used by syslog-ng PE supports only the following character sets:

- Single-byte character sets: `US7ASCII`, `WE8DEC`, `WE8MSWIN1252`, and `WE8ISO8859P1`
- Unicode character sets: `UTF8`, `AL16UTF16`, and `AL32UTF8`

Example: Using the sql() driver with an Oracle database

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

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

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

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

Using the sql() driver with a Microsoft SQL database

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

- The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the syslog-ng PE server. For details, see the following example.
- As certain database versions limit the maximum length of table names, macros in the table names should be used with care.
- In the current version of syslog-ng PE, the types of database columns must be explicitly set for the MSSQL destination.

  CAUTION:

  The following column types cannot be used in MSSQL destinations: `nchar`, `nvarchar`, `ntext`, and `xml`.

- The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng PE. The `varchar` column type can store maximum 4096 bytes-long messages. The maximum length of the messages can be set using the `log-msg-size()` option. For details, see the following example.
- Remote access for SQL users must be explicitly enabled on the Microsoft Windows host running the Microsoft SQL Server. For details, see Configuring Microsoft SQL Server to accept logs from syslog-ng.

Example: Using the sql() driver with an MSSQL database

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

```
destination d_mssql {
    sql(
        type(mssql)
        host("logserver")
        port("1433")
        username("syslogng")
        password("syslogng")
        database("syslogng")
        table("msgs_${R_YEAR}${R_MONTH}${R_DAY}")
```

```
        columns("datetime varchar(16)", "host varchar(32)", "program
varchar(32)", "pid varchar(8)", "message varchar(4096)")
        values("${R_DATE}", "${HOST}", "${PROGRAM}", "${PID}",
"${MSGONLY}")
        indexes("datetime", "host", "program", "pid")
    );
};
```

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

You can copy and customize the following sample:

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

The way syslog-ng PE interacts with the database

SQL operations syslog-ng Premium Edition (syslog-ng PE) uses

Create table:

- If the given table does not exist, syslog-ng PE tries to create it with the given column types.
- The syslog-ng PE application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.
- If syslog-ng PE cannot create or alter a table, it tries to do it again when it reaches the next `time-reopen()`.

Alter table:

- If the table structure is different from given structure in an existing table, syslog-ng PE tries to add columns in this table but never will delete or modify an existing column.
- If syslog-ng PE cannot create or alter a table, it tries to do it again when reach the next `time-reopen()`.
- The syslog-ng PE application requires read and write access to the SQL table, otherwise it cannot verify that the destination table exists.

Insert table:

- Insert new records in a table.
- Inserting the records into the database is performed by a separate thread.
- The syslog-ng PE application automatically performs the escaping required to insert the messages into the database.
- If insert returns with error, syslog-ng PE tries to insert the message +two times by default, then drops it. Retrying time is the value of `time-reopen()`.

Encoding

The syslog-ng PE application uses UTF-8 by default when writes logs into database.

Start/stop and reload

Start:

- The syslog-ng PE application will connect to database automatically after starting regardless existing incoming messages.

Stop:

- The syslog-ng PE application will close the connection to database before shutting down.

Possibility of losing logs:

- The syslog-ng PE application cannot lose logs during shutting down if a disk-buffer file was given and it is not full yet.
- The syslog-ng PE application cannot lose logs during shutting down if no disk-buffer file was given.

Reload:

- The syslog-ng PE application will close the connection to database if it received SIGHUP signal (reload).
- It will reconnect to the database when it tries to send a new message to this database again.

Macros

The value of ${SEQNUM} macro will be overrided by sql driver regardless of local or relayed incoming message.

It will keep growing continuously.

MySQL-specific interaction methods

NOTE: When using the `sql()` destination, consider that syslog-ng Premium Edition (syslog-ng PE) must only connect to fully licensed MySQL databases.

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`.

MsSQL-specific interaction methods

In SQL Server 2005 this restriction is lifted - kind of. The total length of all key columns in an index cannot exceed 900 bytes.

If you are using `null()` in your configuration, be sure that the columns allow `NULL` to insert. Give the column as the following example: `"datetime varchar(16) NULL"`.

The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the syslog-ng server. `[default] date = "%Y-%m-%d %H:%M:%S"`.

sql() destination options

NOTE: To use this destination, syslog-ng Premium Edition (syslog-ng PE) must run in server mode. Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

This driver sends messages into an SQL database. The `sql()` destination has the following options:

columns()

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

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

CAUTION:

The following column types cannot be used in MSSQL destinations: `nchar`, `nvarchar`, `ntext`, and `xml`.

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.

dbd-option()

Type:	string
Default:	empty string

Description: Specify database options that are set whenever syslog-ng PE connects to the database server. Consult the documentation of your database server for details on the available options.

Syntax:

```
dbd-option(OPTION_NAME VALUE)
```

OPTION_NAME is always a string, VALUE is a string or a number. For example:

```
dbd-option("null.sleep.connect" 1)
dbd-option("null.sleep.query" 5)
```

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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:	list of flags
Default:	empty string

Description: Flags related to the `sql()` destination.

- *dont-create-tables*: Enable this flag to prevent syslog-ng PE from creating non-existing database tables automatically. The syslog-ng PE application typically has to create tables if you use macros in the table names. Available in syslog-ng PE version 3.24.0 and later.
- *explicit-commits*: By default, syslog-ng PE commits every log message to the target database individually. When the `explicit-commits` option is enabled, messages are committed in batches. This improves the performance, but results in some latency, as the messages are not immediately sent to the database. The size and frequency of batched commits can be set using the `flush-lines()` and `flush-timeout()` parameters. The `explicit-commits` option is available in syslog-ng PE version 3.24.0 and later.

Example: Setting flags for SQL destinations

The following example sets the `dont-create-tables` and `explicit-commits` flags for an `sql()` destination.

```
flags(dont-create-tables,explicit-commits)
```

flush-lines()

Type:	number
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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, unless you set `ignore-tns-ora(yes)`.

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`.

ignore-tns-ora()

Type:	yes or no
Default:	no

Description: If set to yes, the Oracle sql destination does not use the `/etc/tnsnames.ora` file, but uses the hostname set in the `host()` option. Available in syslog-ng Premium Edition version 7.0.93.16.

indexes()

Type:	string list
Default:	"date", "facility", "host", "program"

Description: The list of columns that are indexed by the database to speed up searching. To disable indexing for the destination, include the empty `indexes()` parameter in the destination, simply omitting the `indexes` parameter will cause syslog-ng to request indexing on the default columns.

The syslog-ng PE application will create the name of indexes automatically with the following method:

- In case of MsSQL, PostgreSQL, MySQL or SQLite or (Oracle but tablename < 30 characters): `{table}_{column}_idx`.
- In case of Oracle and tablename > 30 characters: `md5sum of {table}_{column}-1` and the first character will be replaced by "i" character and the md5sum will be truncated to 30 characters.

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
Default:	Use global setting.

Description: The number of messages that the output queue can store.

null()

Type:	string
Default:	

Description: If the content of a column matches the string specified in the `null()` parameter, the contents of the column will be replaced with an SQL NULL value. If unset (by default), the option does not match on any string. For details, see the Example: Using SQL NULL values.

Example: Using SQL NULL values

The `null()` parameter of the SQL driver can be used to replace the contents of a column with a special SQL NULL value. To replace every column that contains an empty string with NULL, use the `null("")` option, for example

```
destination d_sql {
    sql(
        type(pgsql)
        host("logserver")
        username("syslog-ng")
        password("password")
        database("logs")
```

```
        table("messages_${HOST}_${R_YEAR}${R_MONTH}${R_DAY}")
        columns("datetime", "host", "program", "pid", "message")
        values("${R_DATE}", "${HOST}", "${PROGRAM}", "${PID}",
"${MSGONLY}")
        indexes("datetime", "host", "program", "pid", "message")
        null("")
    );
};
```

To replace only a specific column (for example, `pid`) if it is empty, assign a default value to the column, and use this default value in the `null()` parameter:

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

Ensure that the default value you use does not appear in the actual log messages, because other occurrences of this string will be replaced with NULL as well.

password()

Type:	string
Default:	n/a

Description: Password of the database user.

port()

Type:	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.

retries()

Type:	number (insertion attempts)
Default:	3

Description: The number of insertion attempts. If syslog-ng PE could not insert a message into the database, it will repeat the attempt until the number of attempts reaches `retries`, then drops the connection to the database. For example, syslog-ng PE will try to insert a message maximum three times by default (once for first insertion and twice if the first insertion was failed).

session-statements()

Type:	comma-separated list of SQL statements
Default:	empty string

Description: Specifies one or more SQL-like statement which is executed after syslog-ng PE has successfully connected to the database. For example:

```
session-statements("SET COLLATION_CONNECTION='utf8_general_ci'")
```

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.

table()

Type:	string
Default:	messages

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

time-zone()

Type:	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 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.

type()

Type:	mssql, mysql, oracle, pgsql, or sqlite3
Default:	mysql

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.

username()

Type:	string
Default:	n/a

Description: Name of the database user.

values()

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

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

It is possible to give a special value calling: default (without quotation marks). It means that the value will be used that is the default of the column type of this value.

Example: Value: default

```
columns("date datetime", "host varchar(32)", "row_id serial")
values("${R_DATE}", "${HOST}", default)
```

stackdriver: Sending logs to the Google Stackdriver cloud

The `stackdriver()` destination of syslog-ng PE can send log messages to the Google Stackdriver cloud. Google Stackdriver is a widely used metrics, event, and log aggregator and analyzer system. The `stackdriver` destination is available in syslog-ng PE version 7.0.14 and later.

How the stackdriver() destination works

The `stackdriver()` destination uses the HTTP REST API to perform OAuth2 authentication to Google Stackdriver and obtains an access token from Stackdriver using the key specified in a JSON file. This access token is required to send logs to Stackdriver using the Stackdriver Logging API.

The syslog-ng PE application automatically refreshes the token when it expires (usually every 60 minutes). The syslog-ng PE application stores the token it obtains, even if you restart or reload syslog-ng PE. If you change the JSON key on your syslog-ng PE host, syslog-ng PE will start using the new key only when the stored access token expires or becomes invalid.

By default, syslog-ng PE uses the default system CA certificate store to validate the certificate sent by Google Stackdriver. If the Certificate Authority of the certificate sent by Google Stackdriver is not available on your host, you must download the CA certificate and add it to the certificate store. The location of the certificate store depends on your platform. Most commonly, it is one of the following (or a similar location):

- `/etc/ssl/certs/`
- `/etc/pki/ca-trust/`
- `/etc/pki/tls/certs/`

Limitations

- The `log_id()` option of the destination currently does not support macros or templates, only strings. As a result, every log entry has the same log id.
- Currently the following resource types are supported: `generic_node`, `generic_task`, and `global`. You can configure other resource types, but they are untested.
- Each syslog-ng PE `stackdriver()` destination can use only one resource type. If you want to send logs using multiple resource types, you must configure multiple `stackdriver()` destinations.
- When referring to options in the syslog-ng PE configuration file, the hyphen (-) and underscore (_) characters are usually interchangeable. In the `stackdriver` destination, you must use underscore (_) in the options that syslog-ng PE passes directly to Google Stackdriver. These options are the following:
    - `gcp_auth_header` and its contents
    - `log_id`

- `project_id`
- `resource()` and its contents.

NOTE: Typically, only the central syslog-ng PE server uses this destination. For more information on the server mode, see Server mode.

Declaration

```
destination d_stackdriver {
    stackdriver(
       gcp_auth_header(
          credentials("<path-to-the-service.json>")
       )
       log_id("<folder-name-for-logs-in-stackdriver>")
       resource(
          project_id("<identifier-of-the-GCP-project>")
          <parameters-of-the-monitored-resource>
       )
    );
};
```

Example: Sending log messages to Google Stackdriver

Using a `generic_node()` resource type to send log messages to Google Stackdriver

```
destination d_stackdriver {
    stackdriver(
       gcp_auth_header(
          credentials("/opt/syslog-ng/etc/service.json")
       )
       log_id("123456")
       resource(
          generic_node(
             project_id("my-test-project")
             location("EU/Budapest")
             namespace("my cluster")
             node_id("$HOST")
          )
       )
       [...]
    );
};
```

Batch size

The `batch-lines()`, `batch-lines()`, and `batch-timeout()` options of the destination determine how many log messages syslog-ng PE sends in a batch. The `batch-lines()` option determines the maximum number of messages syslog-ng PE puts in a batch in. This can be limited based on size and time:

- syslog-ng PE sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. This ensures that the destination receives every message in a timely manner even if suddenly there are no more messages.
- syslog-ng PE sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Configuring syslog-ng PE to send logs to Google Stackdriver

This procedure summarizes how to send log messages from syslog-ng Premium Edition (syslog-ng PE) to Google Stackdriver.

Prerequisites

- A valid Google Stackdriver account.
- syslog-ng Premium Edition version 7.0.14 or later.

To send log messages to Google Stackdriver

1. Login to your Google Cloud Stackdriver account.
2. Create a new project and note its Project ID.

   NOTE: If you want to use an existing project, navigate to the project and check its Project ID in the Project info page.
3. Create a service account for the project.
4. Create a key for service account. Download the key as a JSON file.
5. Configure the roles of the service account.. Make sure that the service account has at least the logging.logWriter role.
6. Configure a `stackdriver` destination in syslog-ng Premium Edition. As a minimum, you must set the service account key in the `credentials()` option, the `log_id()` option, and `resource` option (which must include the `project_id()` field). For details on the available options, see stackdriver destination options.

stackdriver destination options

The `stackdriver` destination of syslog-ng PE can send log messages to the Google Stackdriver cloud. The `stackdriver` destination has the following options. Available in syslog-ng PE version 7.0.14 and later.

CAUTION:

When referring to options in the syslog-ng PE configuration file, the hyphen (-) and underscore (_) characters are usually interchangeable. In the `stackdriver` destination, you must use underscore (_) in the options that syslog-ng PE passes directly to Google Stackdriver. These options are the following:

- **`gcp_auth_header` and its contents**
- **`log_id`**
- **`project_id`**
- **`resource()` and its contents.**

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, syslog-ng PE sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in syslog-ng PE version 3.197.0.12 and later.

For more information on how this option influences batch mode, see Batch size on page 604.

batch-lines()

Type:	number [lines]
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The syslog-ng PE application waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

For example, if you set `batch-lines()` to 100, syslog-ng PE waits for 100 messages.

If the `batch-timeout()` option is disabled, the syslog-ng PE application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

If the `batch-timeout()` option is enabled and the queue becomes empty, syslog-ng PE flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the syslog-ng PE source that feeds messages to this destination is configured properly: the value of the `log-iw-size()` option of the source must be higher than the `batch-lines()*workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

For more information on how this option influences batch mode, see Batch size on page 604.

batch-timeout()

Type:	time [milliseconds]
Default:	-1 (disabled)

Description: Specifies the time syslog-ng PE waits for lines to accumulate in the output buffer. The syslog-ng PE application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, syslog-ng PE sends messages to the destination once every `batch-timeout()` milliseconds at most.

For more information on how this option influences batch mode, see Batch size on page 604.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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")
              )
       );
};
```

gcp_auth_header()

Description: This option stores the service key and other parameters needed for the `stackdriver` destination to successfully connect to Google Stackdriver. It has the following options. The `credentials()` option is a required options.

credentials()	
Type:	path
Default:	N/A

Description: The path to the JSON file that contains the key for the service account. The service account must have at least the `logging.write` role. For details, see Configuring syslog-ng PE to send logs to Google Stackdriver

The syslog-ng PE application automatically refreshes the token when it expires (usually every 60 minutes). The syslog-ng PE application stores the token it obtains, even if you restart or reload syslog-ng PE. If you change the JSON key on your syslog-ng PE host, syslog-ng PE will start using the new key only when the stored access token expires or becomes invalid.

hook-commands()

Description: This option makes it possible to execute external programs when the relevant driver is initialized or torn down. The `hook-commands()` can be used with all source and destination drivers with the exception of the `usertty()` and `internal()` drivers.

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 or SELinux, you might have to modify your AppArmor or SELinux configuration to enable syslog-ng PE to execute external applications.

Using the hook-commands() when syslog-ng PE starts or stops

To execute an external program when syslog-ng PE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed when syslog-ng PE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed when syslog-ng PE stops.	

Using the hook-commands() when syslog-ng PE reloads

To execute an external program when the syslog-ng PE configuration is initiated or torn down (for example, on startup/shutdown or during a syslog-ng PE reload), use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng PE configuration is initiated, for example, on startup or during a syslog-ng PE reload.	

teardown()	
Type:	string

Default:	N/A

Description: Defines an external program that is executed when the syslog-ng PE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng PE reload.

Example: Using the `hook-commands()` with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically when syslog-ng PE is started/stopped.

The assumption in this example is that the `LOGCHAIN` chain is part of a larger ruleset that routes traffic to it. Whenever the syslog-ng PE created rule is there, packets can flow (otherwise the port is closed).

```
source  {
    network(transport(udp)
       hook-commands(
          startup("iptables  -I  LOGCHAIN  1  -p  udp  --dport  514  -j
ACCEPT")
          shutdown("iptables  -D  LOGCHAIN  1")
       ;)
    );
 };
```

json-payload()

Accepted values:	template function
Default:	$(format-json --scope rfc5424 --exclude DATE --key ISODATE)

Description: The payload of the log message that syslog-ng PE sends to Google Stackdriver. The payload must be a JSON-formatted list of name-value pairs. For details on selecting name-value pairs, see value-pairs(). By default, syslog-ng PE uses the `rfc5424` scope, which is the same as the one that the `syslog()` destination uses.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

log_id()

Accepted values:	string
Default:	none

Description: The name of the folder that stores the logs in Google Stackdriver. The value of the `log_id()` option must be less than 512 characters long and can only include the following characters: upper and lower case alphanumeric characters, forward-slash, underscore, hyphen, and period.

NOTE: The `log-id()` option is mandatory. If not referred to, the Stackdriver server will send the following HTTP response to syslog-ng PE:

```
"error": {
        "code": 400,
        "message":"Log name contains illegal character",
        "status":"INVALID_ARGUMENT"
```

persist-name()

Type:	string
Default:	None

*Description:*If you receive the following error message during syslog-ng PE startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error occurs if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

The `proxy()` option enables you to configure the `stackdriver` driver to use a specific HTTP proxy for all HTTP-based destinations, instead of using the proxy that is configured for the system.

If you do not set the `proxy()` option, the `stackdriver` driver uses the `http_proxy` and `https_proxy` environment variables, as shown in CURLOPT_PROXY explained.

NOTE: Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.

resource()

Type:	string
Default:	None

Description: Specifies the type of the monitored resource and its fields, as described in the Google Stackdriver documentation. You can use strings, macros, and templates in the values of the fields.

Currently the following resource types are supported: `generic_node`, `generic_task`, and `global`. You can configure other resource types, but they are untested.

One Identity recommends using the generic_node resource type. Each syslog-ng PE `stackdriver` destination can use only one resource type. If you want to send logs using multiple resource types, you must configure multiple `stackdriver` destinations.

CAUTION:

Make sure that you set the resource type and its fields correctly. The value of the `project_id()` field must be the Project ID of your Google Stackdriver project. The syslog-ng PE application cannot check the validity of this option.

For example:

```
resource(
  generic_node(
    project_id("my-test-project")
    location("EU/Budapest")
    namespace("my cluster")
    node_id("$HOST")))
```

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.

The syslog-ng PE application handles HTTP error responses the following way.

- If the server returns the 401 response code because the token expired, syslog-ng PE automatically requests a new token and resends the message.
- If the HTTP server returns 4xx codes, syslog-ng PE will drop the messages.
- If the HTTP server returns 5xx codes syslog-ng PE will attempt to resend messages until the number of attempts reaches `retries`.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

timeout()

Type:	number [seconds]
Default:	0

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded. By default, the timeout value is `0`, meaning that there is no timeout. Available in version 3.117.0.4 and later.

use-system-cert-store()

Type:	yes | no
Default:	no

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that syslog-ng PE uses to send messages to the server. Increasing the number of worker threads can drastically

improve the performance of the destination.

CAUTION:

Hazard of data loss!

When you use more than one worker threads together with the disk-buffer option, syslog-ng PE creates a separate disk-buffer file for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk-buffer files. Do not decrease the number of workers when the disk-buffer files are in use.

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1" "site2" "site3")`), set the `workers()` option to 3 or more.

`syslog`: Sending messages to a remote logserver using the IETF-syslog protocol

The `syslog()` driver sends messages to a remote host (for example, a syslog-ng server or relay) on the local intranet or internet using the new standard syslog protocol developed by IETF (for details about the new protocol, see 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 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 — 514, TLS — 6514.

Example: 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 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')
              )
       );
};
```

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 IETF-syslog messages). The protocol supports sending messages using the 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 file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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()

Description: Available only in syslog-ng Premium Edition version 7.0.93.17 and later. For details about how client-side failover works, see Client-side failover.

servers()

Type:	list of IP addresses and fully-qualified domain names

Default:	empty

Description: Specifies a secondary destination server where log messages are sent if the primary server becomes inaccessible. To list several failover servers, separate the address of the servers with comma. By default, syslog-ng PE waits for the a server before switching to the next failover server is set in the `time-reopen()` option.

If `failback()` is not set, syslog-ng PE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

failback()

Description: Available only in syslog-ng Premium Edition version 7.0.103.17 and later.

When syslog-ng PE starts up, it always connects to the primary server first. In the `failover()` option there is a possibility to customize the failover modes.

Depending on how you set the `failback()` option, syslog-ng PE behaves as follows:

- **round-robin mode**: If `failback()` is not set, syslog-ng PE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

  Example: round-robin mode

  In the following example syslog-ng PE handles the logservers in round-robin fashion if the primary logserver becomes inaccessible (therefore `failback()` option is not set).

  ```
  destination d_network {
       network(
            "primary-server.com"
            port(601)
            failover( servers("failover-server1", "failover-server2")
  )
  );
  };
  ```

- **failback mode**: If `failback()` is set, syslog-ng PE attempts to return to the

primary server.

After syslog-ng PE connects a secondary server during a failover, it sends a probe every `tcp-probe-interval()` seconds towards the primary server. If the primary logserver responds with a TCP ACK packet, the probe is successful. When the number of successful probes reaches the value set in the `successful-probes-required()` option, syslog-ng PE tries to connect the primary server using the last probe.

NOTE: syslog-ng PE always waits for the result of the last probe before sending the next message. So if one connection attempt takes longer than the configured interval, that is, it waits for connection time out, you may experience longer intervals between actual probes.

Example: failback mode

In the following example syslog-ng PE attempts to return to the primary logserver, as set in the `failback()` option: it will check if the server is accessible every `tcp-probe-interval()` seconds, and reconnect to the primary logserver after three successful connection attempts.

```
destination d_network_2 {
     network(
          "primary-server.com"
          port(601)
          failover(
               servers("failover-server1", "failover-server2")
               failback(
                    successful-probes-required()
                    tcp-probe-interval()
               )
          )
);
};
```

Default value for `tcp-probe-interval()`: 60 seconds

Default value for `successful-probes-required()`: 3

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 inaccessible.

Example: 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the

timestamps after 6 digits.

ip-protocol()

Type:	number
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 `ip-protocol(4)`.

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)`. You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source  s_network_tcp  {
       network(
             transport("tcp")
             ip("::")
             ip-protocol(6)
             port(601)
       );
 };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is 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
Default:	0

Description: The port number to bind to. Messages are sent from this port.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of 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.xsyslog-ng PE 3.3 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. Note that setting the global mark-mode() to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 and later.

port() or destport()

Type:	number
Default:	514

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make syslog-ng send messages to a broadcast address. 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
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
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. It is useful when you want to perform some kind of preprocessing via syslog-ng then forward messages to your central log management solution with the source address of the original sender. This option only works for UDP destinations though the original message can be received by TCP as well. This option is only available if syslog-ng was compiled using the `--enable-spoof-source` configuration option.

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the `Last message repeated n times.` message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequential keepalive probes, regardless of the traffic exchanged in the connection. This option is equivalent to `/proc/sys/net/ipv4/tcp_keepalive_intvl`. The default value is `0`, which means using the kernel default.

CAUTION:

The `tcp-keepalive-time()`, `tcp-keepalive-probes()`, and `tcp-keepalive-intvl()` options only work on platforms which support the `TCP_KEEPCNT`, `TCP_KEEPIDLE`,and `TCP_KEEPINTVL` setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after `tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes()` seconds.

Available in syslog-ng PE version 3.4 and later.

tcp-keepalive-probes()

Type:	number
Default:	0

Description: Specifies the number of unacknowledged probes to send before considering the connection dead. This option is equivalent to `/proc/sys/net/ipv4/tcp_keepalive_probes`. The default value is `0`, which means using the kernel default.

CAUTION:

The `tcp-keepalive-time()`, `tcp-keepalive-probes()`, and `tcp-keepalive-intvl()` options only work on platforms which support the `TCP_KEEPCNT`, `TCP_KEEPIDLE`,and `TCP_KEEPINTVL` setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after `tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes()` seconds.

Available in syslog-ng PE version 3.47.0 and later.

tcp-keepalive-time()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (in seconds) between the last data packet sent and the first keepalive probe. This option is equivalent to `/proc/sys/net/ipv4/tcp_keepalive_time`. The default value is `0`, which means using the kernel default.

CAUTION:

The `tcp-keepalive-time()`, `tcp-keepalive-probes()`, and `tcp-keepalive-intvl()` options only work on platforms which support the `TCP_KEEPCNT`, `TCP_KEEPIDLE`,and `TCP_KEEPINTVL` setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after `tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes()` seconds.

Available in syslog-ng PE version 3.4 and later.

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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 TLS options.

transport()

Type:	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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

syslog-ng(): Forward logs to another syslog-ng node

The `syslog-ng()` destination driver forwards log messages to another syslog-ng node in EWMM format.

The Enterprise-wide message model or EWMM allows you to deliver structured messages from the initial receiving syslog-ng component right up to the central log server, through any number of hops. It does not matter if you parse the messages on the client, on a relay, or on the central server, their structured results will be available where you store the messages. Optionally, you can also forward the original raw message as the first syslog-ng component in your infrastructure has received it, which is important if you want to forward a message for example, to a SIEM system. To make use of the enterprise-wide message model, you have to use the syslog-ng() destination on the sender side, and the default-network-drivers() source on the receiver side.

The `syslog-ng()` destination driver is available in version 7.0.93.16 and later. The node that receives this message must use the default-network-drivers() source to properly handle the messages.

The following is a sample log message in EWMM format.

```
<13>1 2018-05-13T13:27:50.993+00:00 my-host @syslog-ng - - -
{"MESSAGE":"<34>Oct 11 22:14:15 mymachine su: 'su root' failed for username on
/dev/pts/8","HOST_FROM":"my-host","HOST":"my-host","FILE_NAME":"/tmp/in","._
TAGS":".source.s_file"}
```

Declaration

```
destination  d_ewmm  {
       syslog-ng(server("192.168.1.1"));
 };
```

Note in this driver you have to set the address of the destination server using the `server()` parameter (in some other destinations, this parameter does not have an explicit name).

syslog-ng() destination options

The `syslog-ng()` destination is a special version of the `network()` destination driver: by default, it sends EWMM-formatted log messages to the TCP514 port of the server.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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()

Description: Available only in syslog-ng Premium Edition version 7.0.93.17 and later. For details about how client-side failover works, see Client-side failover.

servers()	
Type:	list of IP addresses and fully-qualified domain names
Default:	empty

Description: Specifies a secondary destination server where log messages are sent if the primary server becomes inaccessible. To list several failover servers, separate the address of the servers with comma. By default, syslog-ng PE waits for the a server before switching to the next failover server is set in the `time-reopen()` option.

If `failback()` is not set, syslog-ng PE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

failback()

Description: Available only in syslog-ng Premium Edition version 7.0.103.17 and later.

When syslog-ng PE starts up, it always connects to the primary server first. In the `failover()` option there is a possibility to customize the failover modes.

Depending on how you set the `failback()` option, syslog-ng PE behaves as follows:

- **round-robin mode**: If `failback()` is not set, syslog-ng PE does not attempt to

return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

Example: round-robin mode

In the following example syslog-ng PE handles the logservers in round-robin fashion if the primary logserver becomes inaccessible (therefore `failback()` option is not set).

```
destination d_network {
     network(
          "primary-server.com"
          port(601)
          failover( servers("failover-server1", "failover-server2")
)
);
};
```

- **failback mode**: If `failback()` is set, syslog-ng PE attempts to return to the primary server.

  After syslog-ng PE connects a secondary server during a failover, it sends a probe every `tcp-probe-interval()` seconds towards the primary server. If the primary logserver responds with a TCP ACK packet, the probe is successful. When the number of successful probes reaches the value set in the `successful-probes-required()` option, syslog-ng PE tries to connect the primary server using the last probe.

  NOTE: syslog-ng PE always waits for the result of the last probe before sending the next message. So if one connection attempt takes longer than the configured interval, that is, it waits for connection time out, you may experience longer intervals between actual probes.

Example: failback mode

In the following example syslog-ng PE attempts to return to the primary logserver, as set in the `failback()` option: it will check if the server is accessible every `tcp-probe-interval()` seconds, and reconnect to the primary logserver after three successful connection attempts.

```
destination d_network_2 {
```

```
    network(
        "primary-server.com"
        port(601)
        failover(
            servers("failover-server1", "failover-server2")
            failback(
                successful-probes-required()
                tcp-probe-interval()
            )
        )
);
};
```

Default value for `tcp-probe-interval()`: 60 seconds

Default value for `successful-probes-required()`: 3

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 inaccessible.

Example: 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

ip-protocol()

Type:	number
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 `ip-protocol(4)`.

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)`. You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source  s_network_tcp  {
       network(
             transport("tcp")
             ip("::")
             ip-protocol(6)
             port(601)
       );
 };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is 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
Default:	0

Description: The port number to bind to. Messages are sent from this port.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of 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.xsyslog-ng PE 3.3 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. Note that setting the global `mark-mode()` to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 and later.

port() or destport()

Type:	number
Default:	514

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

server()

Type:	hostname or IP address
Default:	127.0.0.1

Description: The hostname or IP address of the syslog-ng server.

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
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
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.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequential keepalive probes, regardless of the traffic exchanged in the connection. This option is equivalent to `/proc/sys/net/ipv4/tcp_keepalive_intvl`. The default value is `0`, which means using the kernel default.

CAUTION:

The `tcp-keepalive-time()`, `tcp-keepalive-probes()`, and `tcp-keepalive-intvl()` options only work on platforms which support the `TCP_KEEPCNT`, `TCP_KEEPIDLE`,and `TCP_KEEPINTVL` setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after `tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes()` seconds.

Available in syslog-ng PE version 3.4 and later.

tcp-keepalive-probes()

Type:	number
Default:	0

Description: Specifies the number of unacknowledged probes to send before considering the connection dead. This option is equivalent to `/proc/sys/net/ipv4/tcp_keepalive_probes`. The default value is `0`, which means using the kernel default.

CAUTION:

The `tcp-keepalive-time()`, `tcp-keepalive-probes()`, and `tcp-keepalive-intvl()` options only work on platforms which support the `TCP_KEEPCNT`, `TCP_KEEPIDLE`,and `TCP_KEEPINTVL` setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after `tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes()` seconds.

Available in syslog-ng PE version 3.47.0 and later.

tcp-keepalive-time()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (in seconds) between the last data packet sent and the first keepalive probe. This option is equivalent to `/proc/sys/net/ipv4/tcp_keepalive_time`. The default value is `0`, which means using the kernel default.

CAUTION:

The `tcp-keepalive-time()`, `tcp-keepalive-probes()`, and `tcp-keepalive-intvl()` options only work on platforms which support the `TCP_KEEPCNT`, `TCP_KEEPIDLE`,and `TCP_KEEPINTVL` setsockopts. Currently, this is Linux.

A connection that has no traffic is closed after `tcp-keepalive-time() + tcp-keepalive-intvl() * tcp-keepalive-probes()` seconds.

Available in syslog-ng PE version 3.4 and later.

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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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 TLS options.

transport()

Type:	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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

`tcp`, `tcp6`, `udp`, `udp6`: 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 network: Collecting messages using the RFC3164 protocol (network() driver) and network: 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 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 network: Collecting messages using the RFC3164 protocol (network() driver) and network: 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 Change an old destination driver to the network() driver.

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 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')
             )
       );
 };
```

`unix-stream`, `unix-dgram`: 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 unix-stream() and unix-dgram() destination options.

Declaration

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

Example: 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:

create-dirs()

Type:	yes or no
Default:	no

Description: Enable creating non-existing directories when creating files or socket files.

disk-buffer()

Description: This option enables putting outgoing messages into the disk-buffer file of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

Note that changing the value the `dir()` option will not move or copy existing files from the old directory to the new one.

CAUTION:

When creating a new `dir()` option for a disk-buffer file, or modifying an existing one, make sure you delete the persist file.

syslog-ng PE creates disk-buffer files based on the path recorded in the persist file. Therefore, if the persist file 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 file 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 file. 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:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer file already exists, the change will take effect when the disk-buffer file becomes empty.

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 option 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 file, the messages stored in the disk-buffer file will be lost.

truncate-size-ratio()

Type:	number (for percentage) between **`0`** and **`1`**
Default:	`0.1` (10%)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down disk I/O operations, but it saves disk space. As a result, syslog-ng PE only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `disk-buf-size()`.

CAUTION:

One Identity recommends that you do not modify the value of the `truncate-size-ratio()` option unless you are fully aware of the potential performance implications.

Example: 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
Default:	Use global setting.

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to a 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() (DEPRECATED)

Type:	time in milliseconds
Default:	Use global setting.

Description: This is a deprecated option. Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

log-fifo-size()

Type:	number
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.

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.xsyslog-ng PE 3.3 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. Note that setting the global `mark-mode()` to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 and later.

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
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
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 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
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using the disk-buffer option 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 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:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format().

usertty: 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: Using the usertty() driver

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

Client-side failover

syslog-ng PE can detect if the remote server of a network destination becomes inaccessible, and start sending messages to a secondary server. You can configure multiple failover servers, so if the secondary server becomes inaccessible as well, syslog-ng PE switches 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.

The primary server is the address you provided in the destination driver configuration and it has a special role. syslog-ng PE nominates this destination over the failover servers, and handles it as the primary address.

When syslog-ng PE starts up, it always connects to the primary server first. In the failover() option there is a possibility to customize the failover modes.

Depending on how you set the failback() option, syslog-ng PE behaves as follows:

- **round-robin mode**: If failback() is not set, syslog-ng PE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng PE attempts to connect the next failover server in the list in round-robin fashion. This is the default behavior in syslog-ng PE version 7.0.9 and earlier.

Example: round-robin mode

In the following example syslog-ng PE handles the logservers in round-robin fashion if the primary logserver becomes inaccessible (therefore `failback()` option is not set).

```
destination  d_network  {
        network(
                "primary-server.com"
                port(601)
                failover(  servers("failover-server1",  "failover-
server2")  )
 );
 };
```

- **failback mode**: If `failback()` is set, syslog-ng PE attempts to return to the primary server.

  After syslog-ng PE connects a secondary server during a failover, it sends a probe every `tcp-probe-interval()` seconds towards the primary server. If the primary logserver responds with a TCP ACK packet, the probe is successful. When the number of successful probes reaches the value set in the `successful-probes-required()` option, syslog-ng PE tries to connect the primary server using the last probe.

  NOTE: syslog-ng PE always waits for the result of the last probe before sending the next message. So if one connection attempt takes longer than the configured interval, that is, it waits for connection time out, you may experience longer intervals between actual probes.

Example: failback mode

In the following example syslog-ng PE attempts to return to the primary logserver, as set in the `failback()` option: it will check if the server is accessible every `tcp-probe-interval()` seconds, and reconnect to the primary logserver after three successful connection attempts.

```
destination d_network_2 {
     network(
          "primary-server.com"
          port(601)
          failover(
               servers("failover-server1", "failover-server2")
               failback(
```

```
                    successful-probes-required()
                    tcp-probe-interval()
                )
            )
);
};
```

If syslog-ng PE is restarted, it attempts to connect the primary server.

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 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 network() destination options and syslog() destination options.

Routing messages: log paths, flags, and filters

Table 11: 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.
drop-unmatched	This flag means that the message is dropped along a log path when it does not match a filter or is discarded by a parser. Without using the `drop-unmatched` flag, syslog-ng PE would continue to process the message along alternative paths.
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: 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: Using log path flags.

Flag	Description
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 Managing incoming and outgoing messages with flow-control.

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: 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.

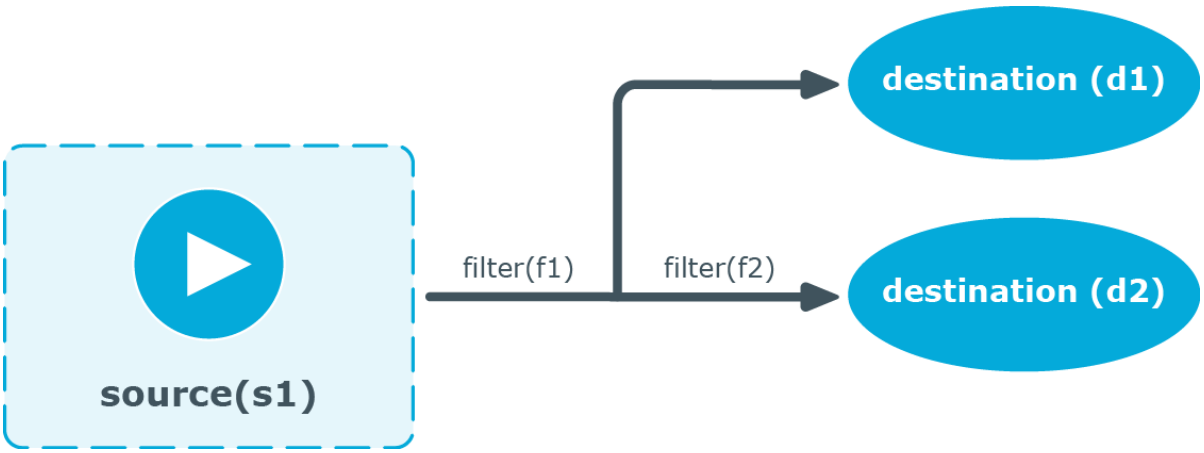
This default behavior can be changed using the `flags()` parameter. Flags apply to individual log paths, they are not global options. For details and examples on the available flags, see Log path flags. The effect and use of the `flow-control` flag is detailed in Managing incoming and outgoing messages 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.

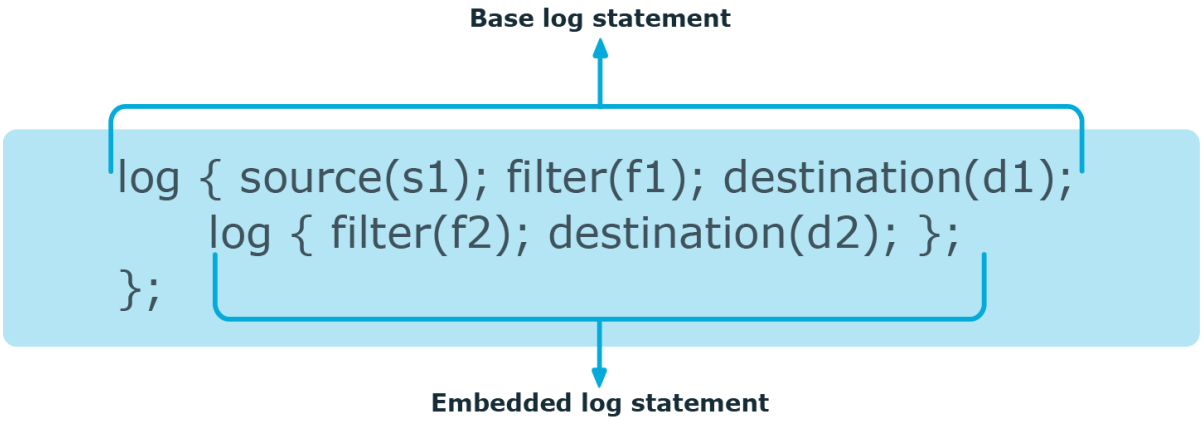
Figure 29: Embedded log statement



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 30: 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 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: 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.

```
log {
    source(s_localhost);
    filter {
        host(192.168.1.1);
    }; destination(d_file1);
```

```
    log {
        message("example");
        destination(d_file2);
    };
};
```

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

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

if-else-elif: Conditional expressions

You can use `if {}`, `elif {}`, and `else {}` blocks to configure conditional expressions.

Conditional expressions have two formats:

- Explicit filter expression:

  ```
  if  (message('foo'))  {
         parser  {  date-parser();  };
  }  else  {
        ...
  };
  ```

  This format only uses the filter expression in `if()`. If `if` does not contain `'foo'`, the `else` branch is taken.

  The `else{}` branch can be empty, you can use it to send the message to the default branch.

- Condition embedded in the log path:

```
if {
      filter { message('foo')); };
      parser { date-parser(); };
} else {
      ...
};
```

This format considers all filters and all parsers as the condition, combined. If the message contains `'foo'` and the `date-parser()` fails, the `else` branch is taken. Similarly, if the message does not contain `'foo'`, the `else` branch is taken.

An alternative, less straightforward way to implement conditional evaluation is to use junctions. For details on junctions and channels, see Junctions and channels.

Junctions and channels

Junctions make it possible to send the messages to different channels, process the messages differently on each channel, and then join every channel together again. You can define any number of channels in a junction: every channel receives a copy of every message that reaches the junction. Every channel can process the messages differently, and at the end of the junction, the processed messages of every channel return to the junction again, where further processing is possible.

A junction includes one or more channels. A channel usually includes at least one filter, though that is not enforced. Otherwise, channels are identical to log statements, and can include any kind of objects, for example, parsers, rewrite rules, destinations, and so on. (For details on using channels, as well as on using channels outside junctions, see Using channels in configuration objects.)

NOTE: Certain parsers can also act as filters:

- The JSON parser automatically discards messages that are not valid JSON messages.
- The `csv-parser()` discards invalid messages if the `flags(drop-invalid)` option is set.

You can also use log-path flags in the channels of the junction. Within the junction, a message is processed by every channel, in the order the channels appear in the configuration file. Typically if your channels have filters, you also set the `flags(final)` option for the channel. However, note that the log-path flags of the channel apply only within the junction, for example, if you set the `final` flag for a channel, then the subsequent channels of the junction will not receive the message, but this does not affect any other log path or junction of the configuration. The only exception is the `flow-control` flag: if you enable flow-control in a junction, it affects the entire log path. For details on log-path flags, see Log path flags.

```
junction {
       channel { <other-syslog-ng-objects> <log-path-flags>};
       channel { <other-syslog-ng-objects> <log-path-flags>};
       ...
};
```

Example: Using junctions

For example, suppose that you have a single network source that receives log messages from different devices, and some devices send messages that are not RFC-compliant (some routers are notorious for that). To solve this problem in earlier versions of syslog-ng PE, you had to create two different network sources using different IP addresses or ports: one that received the RFC-compliant messages, and one that received the improperly formatted messages (for example, using the `flags(no-parse)` option). Using junctions this becomes much more simple: you can use a single network source to receive every message, then use a junction and two channels. The first channel processes the RFC-compliant messages, the second everything else. At the end, every message is stored in a single file. The filters used in the example can be `host()` filters (if you have a list of the IP addresses of the devices sending non-compliant messages), but that depends on your environment.

```
log {
       source { syslog(ip(10.1.2.3) transport("tcp") flags(no-parse));
};
       junction {
              channel { filter(f_compliant_hosts); parser { syslog-parser
(); }; };
              channel { filter(f_noncompliant_hosts); };
       };
       destination { file("/var/log/messages"); };
};
```

Since every channel receives every message that reaches the junction, use the `flags(final)` option in the channels to avoid the unnecessary processing the messages multiple times:

```
log {
    source { syslog(ip(10.1.2.3) transport("tcp") flags(no-parse)); };
    junction {
        channel { filter(f_compliant_hosts); parser { syslog-parser(); };
```

```
flags(final); };
        channel { filter(f_noncompliant_hosts); flags(final); };
    };
    destination { file("/var/log/messages"); };
};
```

Note that syslog-ng PE has several parsers that you can use to parse non-compliant messages. You can even write a custom syslog-ng parser in Python. For details, see parser: Parse and segment structured messages.

NOTE: Junctions differ from embedded log statements, because embedded log statements are like branches: they split the flow of messages into separate paths, and the different paths do not meet again. Messages processed on different embedded log statements cannot be combined together for further processing. However, junctions split the messages to channels, then combine the channels together.

An alternative, more straightforward way to implement conditional evaluation is to configure conditional expressions using `if {}`, `elif {}`, and `else {}` blocks. For details, see if-else-elif: Conditional expressions.

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 Log paths.

Table 12: 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.
drop-unmatched	This flag means that the message is dropped along a log path when it does not match a filter or is discarded by a parser. Without using the `drop-unmatched` flag, syslog-ng PE would continue to process the message along alternative paths.
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:

Flag	Description
	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: 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 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: 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);
};
```

Example: Using the `drop-unmatched` flag

In the following example, if a log message arrives whose `$MSG` part does not contain the string `foo`, then syslog-ng PE will discard the message and will not check compliance with the second `if` condition.

```
...
if {
      filter { message('foo') };
      flags(drop-unmatched)
};
if {
      filter { message('bar') };
};
...
```

(Without the `drop-unmatched` flag, syslog-ng PE would check if the message complies with the second `if` condition, that is, whether or not the message contains the string `bar` .)

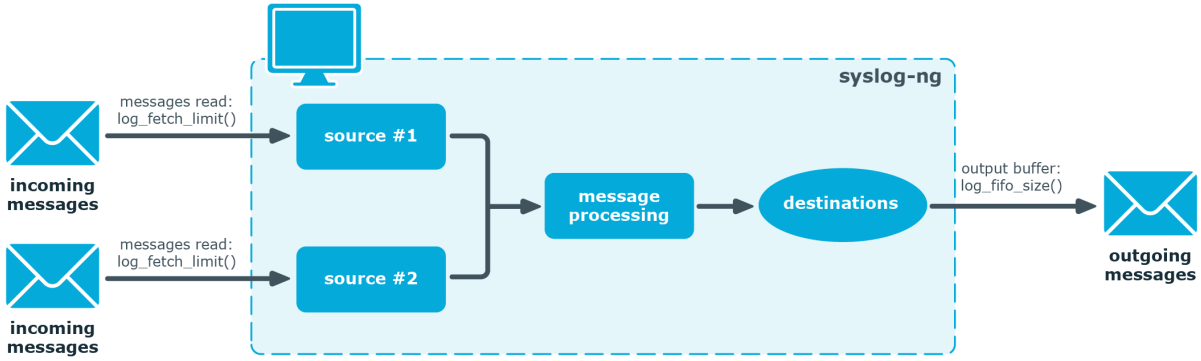
Managing incoming and outgoing messages with flow-control

This section describes the internal message-processing model of syslog-ng Premium Edition (syslog-ng PE), 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 PE 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 PE polls every source and reads the available messages. These messages are processed and put into the output buffer of syslog-ng PE (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 PE 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.

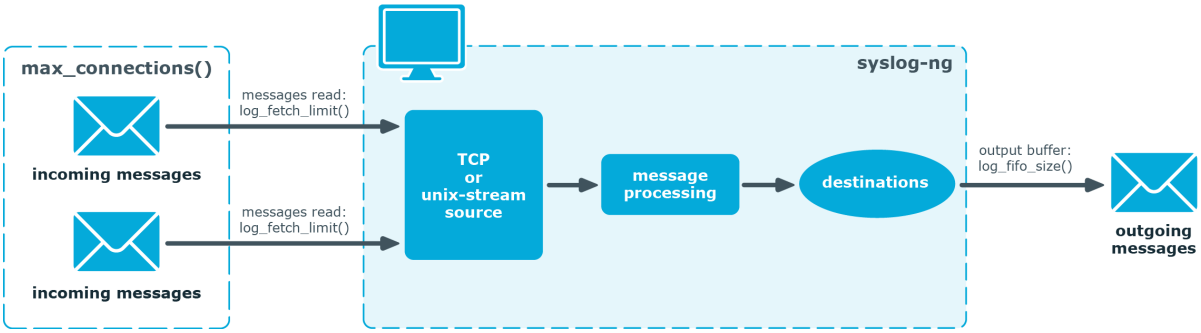
Figure 31: Managing log messages in syslog-ng PE



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 PE reads messages from every connection, thus the `log-fetch-limit()` parameter applies individually to every connection of the source.

Figure 32: Managing log messages of TCP sources in syslog-ng PE



The flow-control of syslog-ng PE introduces a control window to the source that tracks how many messages syslog-ng PE can accept from the source. Every message that syslog-ng PE reads from the source lowers the window size by one, and every message that syslog-ng PE 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 PE suspends the affected sources on the log path.

NOTE: Consider the following regarding suspended sources:

- Although syslog-ng PE suspends sources, the underlying issue is generally in connection with a destination on the log path. Usually, when the issue in connection

with the destination is resolved and syslog-ng PE can send logs again, the window size is adjusted and the source site will function normally.

- When the source is in a suspended state, syslog-ng PE ignores all events related to the affected source, including any connections that may have been terminated since the connection was established. As a result, the commonly used network tools may not list potentially terminated connections as active, but syslog-ng PE tracks them as active connections. The syslog-ng PE application will only detect the connection's actual state when the sources are resumed from their suspended state and syslog-ng PE can start reading from them again.

  The `connections` statistics counter may help you identify when the connections' state is tracked differently by commonly used network tools and by syslog-ng PE.

The initial size of the control window is `100` by default: 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.

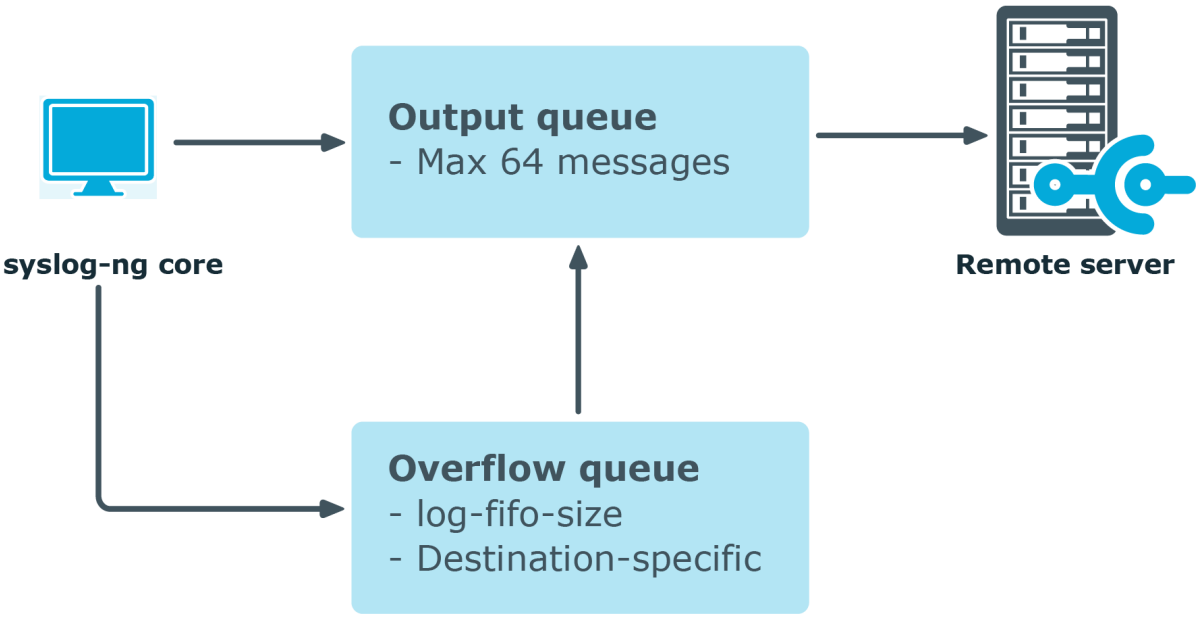
NOTE: If the source can handle multiple connections (for example, `network()`), the size of the control window is divided by the value of the `max-connections()` parameter and this smaller control window is applied to each connection of the source.

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 PE does not activate the flow-control, and messages may be lost.

NOTE:Although syslog-ng PE suspends sources, the underlying issue is generally in connection with a destination on the log path. Usually, when the issue in connection with the destination is resolved and syslog-ng PE can send logs again, the window size increases and the source site will function normally again.

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

Figure 33: Handling outgoing messages in syslog-ng PE



- *Output queue*: Messages from the output queue are sent to the target syslog-ng PE server. The syslog-ng PE 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.
- *The disk-buffer option*: If the output queue is full and the disk-buffer option is enabled, syslog-ng PE puts the outgoing messages into the disk-buffer file of the destination.
- *Overflow queue*: If the output queue is full and the disk-buffer option is disabled (or the disk-buffer file is 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 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: 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: 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); };
```

Configuring flow-control

For details on how flow-control works, see 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 the disk-buffer option, nor flow-control is flow-control is not 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: 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.

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 the disk-buffer option 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 file 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: The disk-buffer option can be used in conjunction with flow-control. For details on flow-control, see Managing incoming and outgoing messages with flow-control.

The following destination drivers can use the disk-buffer option: `amqp()`, `elasticsearch2()`, `file()`, `hdfs()`, `http()`, `kafka()`, `mongodb()`, `program()`, `redis()`, `riemann()`, `sentinel()`, `smtp()`,`sql()`, `stomp()`, `unix-dgram()`, and `unix-stream()`. The `network()`, `syslog()`, `tcp()`, and `tcp6()` destination drivers can also use the disk-buffer option, except when using the `udp` transport method. (The other destinations or protocols do not provide the necessary feedback mechanisms required for the disk-buffer option.)

Every such destination uses a separate disk-buffer file (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-buffer files even when the disk buffers are full.

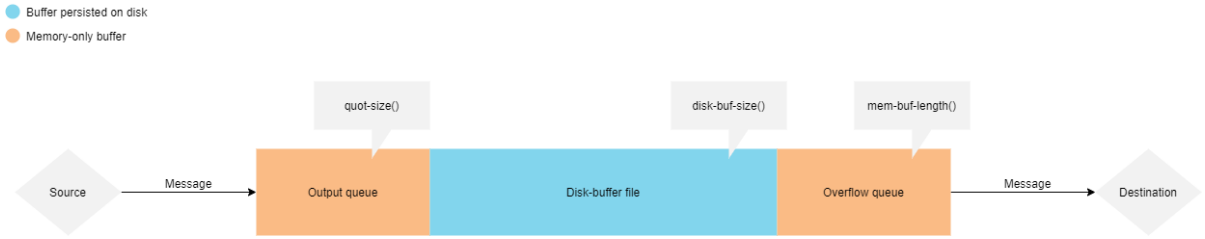
If syslog-ng PE is restarted (using the `/etc/init.d/syslog-ng restart` command, or another appropriate command on your platform), it automatically saves any unsent messages from the disk-buffer file and in-memory queues. After the restart, syslog-ng PE sends the saved messages to the destination. In other words, the disk-buffer file is persistent. The disk-buffer file is also resistant to syslog-ng PE crashes.

The syslog-ng PE application supports two types of disk-buffer options: reliable and normal. For details, see Enabling the reliable disk-buffer option and Enabling the normal disk-buffer option, respectively.

Message handling and the normal disk-buffer option

When you use the disk-buffer option, and the `reliable()` option is set to `no`, syslog-ng PE handles outgoing messages the following way:

Figure 34: Handling outgoing messages in syslog-ng PE with the normal disk-buffer option



- *Output queue*: In-memory queue. If there is space left in it, syslog-ng PE puts the message into this queue first . Messages stored here are processed faster, because syslog-ng PE can skip writing to, and reading from the disk, as well as serializing or deserializing the message, saving I/O and processor time as a result. The contents of the in-memory output queue are persisted to the disk-buffer file during syslog-ng PE reload, restart or stop, but they cannot be persisted if in the event of power failures, or if syslog-ng PE crashes. By default, the output queue can hold 1000 messages (you can adjust this number using the `quot-size()` option).
- *Disk-buffer file*: Disk queue. If there is no space left in the output queue, the message is stored on the disk-buffer file. Messages stored here are persisted on the disk, even in case of power failures or if syslog-ng PE crashes. Using the disk-buffer file takes considerable amount of disk I/O and processor time. The size of this queue can be set with the `disk-buf-size()` option.
- *Overflow queue*: In-memory queue. This queue is used to trigger flow-control if it is set. The contents of the in-memory overflow queue are persisted to the disk-buffer file in case of syslog-ng PE reload, restart or stop, but they are not persisted in case of power failures or if syslog-ng PE crashes. Setting the size of the overflow queue can be done with the `mem-buf-length()` option.

CAUTION:

Hazard of data loss!

In case of normal disk-buffers, the messages stored in the output queue and the overflow queue can be lost in case of power failures or if syslog-ng PE crashes.

CAUTION:

Hazard of data loss!

The syslog-ng PE application does not support storing the disk-buffer files on any kind of network share (that is, NFS, CIFS, and so on). To avoid crashes, data corruption, or data loss, One Identity does not recommend storing your disk-buffer files on network share.

NOTE: Using the disk-buffer option can significantly decrease performance.

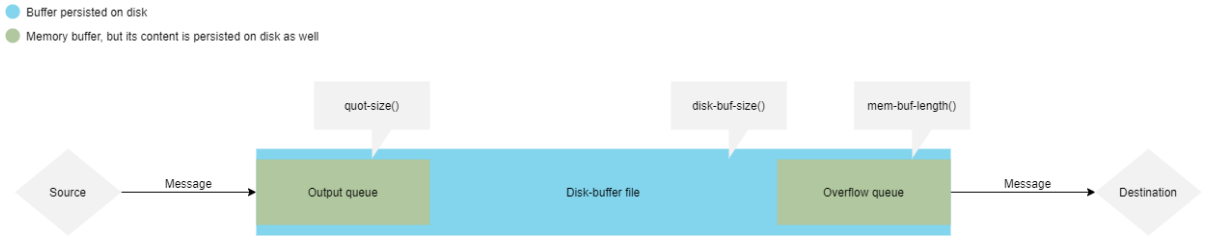
Message handling and using the reliable disk-buffer option

When you use the disk-buffer option, 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. After the size of the disk-buffer file reaches (`disk-buf-size()` minus `mem-buf-size()`), messages are written into both the disk-buffer file and the overflow queue, 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()` times the average message size.)

Figure 35: Handling outgoing messages in syslog-ng PE with the reliable disk-buffer option



- *Output queue*: In-memory and disk queue. If there is space left in it, syslog-ng PE puts the message into this queue first. In case of reliable disk-buffer, in addition to storing the message in memory, it is stored directly in the disk-buffer file as well for safety reasons (see the next point). Messages stored here are processed faster, because syslog-ng PE can skip reading from the disk, and deserializing the message, saving I/O and processor time. By default, the output queue can hold 1000 messages

(you can adjust it using the `quot-size()` option).

- *Disk-buffer file*: Disk queue. If there is no space left in the output queue, the message is stored on the disk-buffer file. Messages stored here are persisted on the disk, and survive syslog-ng PE crash or power failure. Using the disk-buffer file takes considerable amount of disk I/O and processor time. The size of this queue can be set with the `disk-buf-size()` option.
- *Overflow queue*: In-memory and disk queue. This queue is used to trigger flow-control if it is set. Similarly to the output queue, in case of reliable disk-buffer in addition to storing the message in memory, it is stored directly in the disk-buffer file as well for safety reasons. Setting the size of the overflow queue can be done with the `mem-buf-size()` option.

NOTE: When using reliable disk-buffering, the `quot-size()` option sets the number of messages stored in the memory in addition to storing them on the disk. For performance considerations, One Identity recommends that you keep this option set to the default value of `1000`.

Enabling the reliable disk-buffer option

The following destination drivers can use the disk-buffer option: `amqp()`, `elasticsearch2()`, `file()`, `hdfs()`, `http()`, `kafka()`, `mongodb()`, `program()`, `redis()`, `riemann()`, `sentinel()`, `smtp()`,`sql()`, `stomp()`, `unix-dgram()`, and `unix-stream()`. The `network()`, `syslog()`, `tcp()`, and `tcp6()` destination drivers can also use the disk-buffer option, except when using the `udp` transport method. (The other destinations or protocols do not provide the necessary feedback mechanisms required for the disk-buffer option.)

To enable using the reliable disk-buffer option, use the `disk-buffer(reliable(yes))` parameter in the destination. Use the reliable disk-buffer option if you do not want to lose logs in case of reload/restart, an 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: Example for using the reliable disk-buffer option

```
destination d_BSD {
    network(
            "127.0.0.1"
            port(3333)
            disk-buffer(
                mem-buf-size(10000)
```

```
                disk-buf-size(2000000)
                reliable(yes)
            )
        );
};
```

For more details on the differences between the normal and the reliable disk-buffer options, see About disk queue files.

Enabling the normal disk-buffer option

The following destination drivers can use the disk-buffer option: `amqp()`, `elasticsearch2()`, `file()`, `hdfs()`, `http()`, `kafka()`, `mongodb()`, `program()`, `redis()`, `riemann()`, `sentinel()`, `smtp()`,`sql()`, `stomp()`, `unix-dgram()`, and `unix-stream()`. The `network()`, `syslog()`, `tcp()`, and `tcp6()` destination drivers can also use the disk-buffer option, except when using the `udp` transport method. (The other destinations or protocols do not provide the necessary feedback mechanisms required for the disk-buffer option.)

If the `reliable()` option is not set, by default a normal disk-buffer is created. To explicitly enable the normal disk-buffer option, use the `disk-buffer(reliable(no))` parameter in the destination. Use the normal disk-buffer option if you want a solution that is faster than the reliable disk-buffer option. In this case, the process 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`.

Example: Example for using the normal disk-buffer option

When using the plugin for the disk-buffer file

```
destination d_BSD {
    network(
            "127.0.0.1"
            port(3333)
            disk-buffer(
                mem-buf-length(10000)
                disk-buf-size(2000000)
                reliable(no)
            )
        );
        };
```

For more details on the differences between the normal and the reliable disk-buffer options, see About disk queue files.

How to get information about disk-buffer files

This section describes how to get information about disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

NOTE: While reading this section, consider that the default installation path used in the commands and syslog-ng PE files is `/opt/syslog-ng`.

Information about disk-buffer files

This section describes information about disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

The following list contains information about how disk-buffer files are used in syslog-ng PE :

- You can configure `disk-buffer()` for a remote destination in the `destination()` statement.

  For more information about an example of configuring `disk-buffer()` for a remote destination in the `destination()` statement, see disk-buffer().
- By default, syslog-ng PE creates disk-buffer files under `/opt/syslog-ng/var` directory, unless `dir()` option is set in `disk-buffer()`.
- The filenames are generated automatically by syslog-ng PE with the extensions `.qf` for a normal disk-buffer and `.rqf` for a reliable disk-buffer.
- The disk-buffer file stores processed log messages in the format in which they would have been sent out to the destination, but doesn't store information about the destination.

Getting the list of disk-buffer files

This section describes getting the list of disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

The syslog-ng PE application stores information (namely, the `IP:PORT` or `DNS:PORT` of the destinations, and the name of the disk-buffer file) about disk-buffer files in its persist file.

Example: command for listing the disk-buffer files in use

The following command will list the disk-buffer files in use:

```
/opt/syslog-ng/bin/persist-tool dump /opt/syslog-ng/var/syslog-ng.persist
| awk -F '["=]' '/(qfile\(|\.queue)/ { gsub(/[ \t]+/, "", $5); gsub(/^[0-
9A-Fa-f]{8}/, "", $5); "echo "$5"|xxd -r -p"|& getline QUEUE; printf("%s
==> %s\n",$1,QUEUE)}'
```

The example output will look like the following:

```
afsocket_dd_qfile(stream,10.21.10.20:601)  ==> /opt/syslog-ng/var/syslog-
ng-00000.rqf
```

NOTE: If you receive the following error message instead of the example output, install a vim-common package on your system:

```
xxd: command not found
```

Getting the status information of disk-buffer files

This section describes getting the status information of the disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

Command syntax

The basic command syntax for getting the status information of the disk-buffer files used in syslog-ng PE looks like the following:

```
/opt/syslog-ng/bin/dqtool info DISK-BUFFER_FILE
```

Example commands

The following example commands describe how you can get the status information of two different types of disk-buffer files (namely, empty normal disk-buffer files, and non-empty reliable disk-buffer queue files).

Example commands for empty, normal disk-buffer files, and non-empty, reliable disk-buffer queue files

- Empty, normal disk-buffer file

  ```
  /opt/syslog-ng/bin/dqtool info /opt/syslog-ng/var/syslog-ng-00000.qf
  Disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-00000.qf', number_of_messages='0'
  ```

- Non-empty, reliable disk-buffer queue file

  ```
  /opt/syslog-ng/bin/dqtool info /opt/syslog-ng/var/syslog-ng-00000.rqf
  Reliable disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-00000.rqf', number_of_messages='10'
  ```

One-liner command to get the state of disk-buffer files in the default directory

You can use the following one-liner command to get the state of disk-buffer files in the default directory:

```
for qfile in /opt/syslog-ng/var/*.?(r)qf ; do /opt/syslog-ng/bin/dqtool info $qfile 2>&1 ; done
```

Printing the content of disk-buffer files

This section describes printing the content of the disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

Command syntax

The command syntax for printing the content of the disk-buffer files used in syslog-ng PE looks like the following:

```
/opt/syslog-ng/bin/dqtool cat DISK-BUFFER_FILE
```

Short example output for printed content

Example: short output that shows the printed content of the disk-buffer files used in syslog-ng PE

The following short output example shows the printed content of the disk-buffer files used in syslog-ng PE :

```
/opt/syslog-ng/bin/dqtool cat /opt/syslog-ng/var/syslog-ng-00000.rqf

 Reliable disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-00000.rqf', queue_length='2952', size='-437712'
 Jul 31 12:33:48.226 10.21.10.10 <382019-07-31T12:33:36 localhost prg00000[1234]: seq: 0000000838, thread: 0000, runid: 1564569216, stamp: 2019-07-31T12:33:36 PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
 ...
```

Orphan disk-buffer files

This section describes orphan disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

Orphan disk-buffer files

In certain situations (for example, after modifying the disk-buffer configuration or losing the persist information), syslog-ng PE creates a new disk-buffer file instead of using the already existing one. In these situations, the already existing disk-buffer file becomes a so-called orphan disk-buffer file.

> NOTE: The syslog-ng PE application does not store messages in orphan disk-buffer files or forward the messages stored in the disk-buffer file.

Discovering the new disk-buffer files (orphan disk-buffer files)

To discover orphan disk-buffer files, get the list of disk-buffer files from the persist file, then compare the list with the contents of the disk-buffer files' saving directory.

For more information about how you can get the list of disk-buffer files from the persist file, see Getting the list of disk-buffer files).

Example: difference between the list of disk-buffer files from the persist file and the content of the disk-buffer files' saving directory

The following examples show the difference between the list of disk-buffer files from the persist file and the content of the disk-buffer files' saving directory.

Disk-buffer file list from persist file:

```
afsocket_dd_qfile(stream,10.21.10.112:514) = { "queue_file": "/opt/syslog-ng/var/syslog-ng-00001.rqf" }
```

Disk-buffer files' saving directory content:

```
# ls -l /opt/syslog-ng/var/*qf
-rw------- 1 root root 2986780 Jul 31 12:30 /opt/syslog-ng/var/syslog-ng-00000.qf
-rw------- 1 root root 2000080 Jul 31 12:31 /opt/syslog-ng/var/syslog-ng-00000.rqf
-rw------- 1 root root    4096 Aug  1 11:09 /opt/syslog-ng/var/syslog-ng-00001.rqf
```

The disk-buffer files `syslog-ng-00000.qf` and `syslog-ng-00000.rqf` don't exist in the persist file. These two files are the orphan disk-buffer files.

For more information about orphan disk-buffer files and how to process the messages in orphan disk-buffer files using a separate syslog-ng PE instance, see How to process messages from an orphan disk-buffer file using a separate syslog-ng PE instance.

How to process messages from an orphan disk-buffer file using a separate syslog-ng PE instance

This section describes how to read messages from an orphan disk-buffer file by using a separate syslog-ng Premium Edition (syslog-ng PE) process running parallel to the already running syslog-ng PE instance.

Orphan disk-buffer files

In certain situations (for example, after modifying the disk-buffer configuration or losing the persist information), syslog-ng PE creates a new disk-buffer file instead of using the already existing one. In these situations, the already existing disk-buffer file becomes a so-called orphan disk-buffer file.

NOTE: The syslog-ng PE application does not store messages in orphan disk-buffer files or forward the messages stored in the disk-buffer file.

Processing the messages from an orphan disk-buffer file by using a separate syslog-ng PE instance

When syslog-ng PE creates orphan disk-buffer files, you can start a separate syslog-ng PE instance parallel to the syslog-ng PE instance already running, and use the following resolution process to process the messages in the orphan disk-buffer file.

CAUTION:

Before starting a separate syslog-ng PE instance to process the messages from the orphan disk-buffer file, consider the following:

- **During the resolution process, a separate syslog-ng PE instance will be started with its temporary files beside the syslog-ng PE instance already running.**
- **An incorrect startup command and incorrect configurations may cause issues for the syslog-ng PE instance already running.**
- **The disk-buffer file stores processed log messages in the format in which they would have been sent out to the destination.**
- **The disk-buffer file doesn't store information about the destination.**

To process the messages from an orphan disk-buffer file using a separate syslog-ng PE instance,

1. Identify the orphan disk-buffer files and make a record of them. For more information, see How to get information about disk-buffer files.

   It is important to know the type of the disk-buffer file. Disk-buffer file types can be normal (`.qf`) or reliable (`.rqf`).

   In the examples during this process, the `/opt/syslog-ng/var/syslog-ng-00005.rqf` orphan reliable disk-buffer file is used.

2. Determine the destination of the logs. The content of the disk-buffer may help you determine the logs' destination. For more information, see How to get information about disk-buffer files.

   In the examples during this process, the destination `10.21.10.20` is used with the standard `network()` port `514`.

3. Create a directory for the temporary instance. In the examples during this process, the `/tmp/qdisk` directory is used.

   ```
   mkdir /tmp/qdisk
   ```

   CAUTION:

   Make sure that there is sufficient disk space in the directory. The minimum recommended disk space in the directory is equal to the size of the orphan disk-buffer file.

If you want to use a different temporary directory (that is, other than `/tmp/qdisk`), create a symbolic link between `/tmp/qdisk` and the temporary directory you want to use with `ln -s /path/to/tempdir /tmp/qdisk`. This will allow you to use the commands in this resolution process.

If you will not use a different temporary directory, use the `/tmp/qdisk` temporary directory in the example commands and file names.

4. Create the configuration file `/tmp/qdisk/qdisk.conf` for the temporary instance with the following content.

Example: creating the `/tmp/qdisk/qdisk.conf` configuration file for the temporary instance

```
@version:7.0
@include "scl.conf"

options {
   keep-hostname(yes);
   keep-timestamp(yes);
};

destination d_destination {
#      ADD YOUR DESTINATION HERE

};

log {
   destination(d_destination);
};
```

5. Add your destination statement with `disk-buffer()` to the configuration file. You can copy the destination statement from your running syslog-ng PE configuration.

CAUTION:

Add the `dir()` option and set the disk-buffer file's destination directory to the temporary directory (that is, `/tmp/qdisk`) in your destination statement.

Example: adding the destination statement with `disk-buffer()` to the configuration file

```
network("10.21.10.20"
       disk-buffer(
             disk-buf-size(1048576)
             reliable(yes)
             dir(/tmp/qdisk/)
 );
```

6. Start the temporary syslog-ng PE instance in the foreground.

```
syslog-ng -Fe -f /tmp/qdisk/qdisk.conf -R /tmp/qdisk/qdisk.persist -c
/tmp/qdisk/qdisk.ctl
```

The syslog-ng PE application will log to the console, so you will see any potential error that may occur during startup.

The following example output displays that an empty disk-buffer file has been created and the connection to the remote destination has been established.

Example: output displaying newly created empty disk-buffer file and connection established to remote destination

```
Follow-mode file source not found, deferring open; filename='/no_
such_file_or.dir'
Reliable disk-buffer state saved; filename='/tmp/qdisk/syslog-ng-
00000.rqf', qdisk_length='0'
No server license found, running in client mode;
syslog-ng starting up; version='7.0.20', cfg-
fingerprint='eaa03b9efb88b87d7c1b0ce7efd042ed8ac0c013', cfg-nonce-
ndx='0', cfg-signature='c0327a7f7e6418ce0399a75089377dfb662bb072'
FIPS information; FIPS-mode='disabled'
Syslog connection established; fd='7', server='AF_INET
(10.21.10.20:514)', local='AF_INET(0.0.0.0:0)'
```

7. To stop syslog-ng PE , press `CTRL+C`.
8. Overwrite the empty disk-buffer file with the orphan disk-buffer file.

```
mv /opt/syslog-ng/var/syslog-ng-00005.rqf /tmp/qdisk/syslog-ng-00000.rqf
```

9. Start syslog-ng PE using the command used in Start the temporary syslog-ng PE instance in the foreground step.

```
syslog-ng -Fe -f /tmp/qdisk/qdisk.conf -R /tmp/qdisk/qdisk.persist -c
/tmp/qdisk/qdisk.ctl
```

10. Open another terminal and check the progress by using one of the following methods.
    - Checking the number of stored logs in the disk-buffer (that is, the last number from the output).

      ```
      /opt/syslog-ng/sbin/syslog-ng-ctl stats -c /tmp/qdisk/qdisk.ctl |
      grep 'dst.*queued'
      ```

    - Checking the status of the disk-buffer file.

      ```
      /opt/syslog-ng/bin/dqtool info /tmp/qdisk/syslog-ng-00000.rqf
      ```

      An empty disk-buffer file will look similar to this:

      Example: empty disk-buffer file status message

      When checking the status of the disk-buffer files, the terminal will display a similar status message for an empty disk-buffer file:

      ```
      Reliable disk-buffer state loaded; filename='/tmp/qdisk/syslog-
      ng-00000.rqf', queue_length='0', size='0'
      ```

11. Press `CTRL+C` to stop syslog-ng PE .
12. Check the state of the orphan disk-buffer file. For more information, see How to get information about disk-buffer files.
13. If you have more than one orphan disk-buffer file, repeat the steps following the syslog-ng PE stop (that is, the steps beginning from overwriting the empty disk-buffer file with the orphan disk-buffer file) for each orphan disk-buffer file.
14. Remove the temporary directory.

Example: command for removing the temporary directory

The following command removes the `/mp/qdisk` temporary directory:

```
rm -rf /tmp/qdisk
```

How to empty disk-buffer files

This section describes how to empty disk-buffer files used in syslog-ng Premium Edition (syslog-ng PE).

CAUTION:

Hazard of data loss!

You must stop log reception to be able to empty a disk-buffer. If you fail to stop log reception before emptying a disk-buffer, your newly received log messages may get stored in the disk-buffer, overwriting your previous log messages. To avoid log loss, One Identity recommends that you redirect your logs to a different syslog server when emptying your disk-buffer files.

NOTE: Consider the following while reading this section:

This section uses a simple example configuration with one source and one destination with disk-buffer.

If you are not aware of disk-buffers or you're not sure which of your destinations use disk-buffer, One Identity recommends that you do not proceed with the procedure of emptying your disk-buffer files. Instead, One Identity recommends that you contact our Support Team and open a service request. When opening the service request, describe your issue and attach a collected debug bundle from your system.

For more information about collecting a debug bundle for Microsoft Windows, see How to create a syslog-ng debug bundle archive on Windows operating system.

For more information about collecting a debug bundle for Linux or Unix OS, see How to create a syslog-ng debug bundle on Linux Or Unix operating system.

Recommendation

One Identity recommends that you empty your disk-buffer files before you begin the following:

- Upgrading syslog-ng Premium Edition (syslog-ng PE) from version 6 to 7.
- Changing the configuration of a remote destination with disk-buffer.
- Applying a solution that includes the removal of the syslog-ng PE persistent file.

Example configuration for emptying disk-buffer files

The syslog-ng PE application uses the following example configuration to describe how to empty disk-buffer files:

```
source s_net {
       network();
};
destination d_logserver {
       network("10.21.10.20" port(514) disk-buffer( disk-buf-size
(2000000) ) );
};
log {
       source(s_net);
       destination(d_logserver);
};
```

To empty disk-buffer files,

1. Name the disk-buffer file to empty and the destination statement using it.

   If you are not sure about which disk-buffer file to empty, or the destination statement using the disk-buffer file in question, you can use one of the following methods:

   - Check the list and the status of the disk-buffer files.

     Examples

     - Non-empty disk-buffer file

       ```
       Disk-buffer state loaded; filename='/opt/syslog-
       ng/var/syslog-ng-00000.qf', qout_length='0', qbacklog_
       length='0', qoverflow_length='0', qdisk_length='3006'
       ```

     - IP:PORT information of the destination with the disk-buffer in use

       ```
       afsocket_dd_qfile(stream,10.21.10.20:514) = { "queue_
       file": "/opt/syslog-ng/var/syslog-ng-00000.qf" }
       ```

     For more information about getting information about disk-buffer files, see Information about disk-buffer files.

   - Find the destination statement in the syslog-ng PE configuration using the IP:PORT information.

```
destination d_logserver { network("10.21.10.20" port(514) disk-buffer
( disk-buf-size(2000000) ) ); };
```

2. Locate the log statements that use the destination statement you named previously.
3. Disable the sources in the log statements.

   Add '#' at the beginning of all `source()` entries in the log paths.

   ```
   log  {
    #source(s_net);
      destination(d_logserver);
    }
   ```

4. Reload syslog-ng PE by entering the `/opt/syslog-ng/sbin/syslog-ng-ctl reload` command.
5. Check the disk-buffer file status.

   For more information, see Getting the status information of disk-buffer files.
6. To enable the sources again, remove '#' from the log paths and reload syslog-ng PE .

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 the disk-buffer option is not available, 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 the disk-buffer option. Instead, logs are stored only in the memory.

Example: Example for using memory buffering

```
destination  d_BSD  {
       network(
                    "127.0.0.1"
                    port(3333)
                    log-fifo-size(10000)
              );
 };
```

About disk queue files

Normal and reliable queue files

The key difference between disk queue files that employ the `reliable(yes)` option and not is the strategy they employ. Reliable disk queues guarantee that all the messages passing through them are written to disk first, and removed from the queue only after the destination has confirmed that the message has been successfully received. This prevents message loss, for example, due to syslog-ng PE crashes if the client and the destination server communicate using the Advanced Log Transfer Protocol (ALTP). Note that the Reliable Log Transfer Protocol is available only in syslog-ng Premium Edition version 6 LTSsyslog-ng Premium Edition version 6 LTS. Of course, using the `reliable(yes)` option introduces a significant performance penalty as well.

Both reliable and normal disk-buffers employ an in-memory output queue (set in `quot-size()`) and an in-memory overflow queue (set in `mem-buf-size()` for reliable disk-buffers, or `mem-buf-length()` for normal disk-buffers). The difference between reliable and normal disk-buffers is that when the reliable disk-buffer uses one of its in-memory queues, it also stores the message on the disk, whereas the normal disk-buffer stores the message only in memory. The normal disk-buffer only uses the disk if the in-memory output buffer is filled up completely. This approach has better performance (due to fewer disk I/O operations), but also carries the risk of losing a maximum of `quot-size()` plus `mem-buf-length()` number of messages in case of an unexpected power failure or application crash.

Size of the queue files

Disk queue files tend to grow. Each may take up to `disk-buf-size()` bytes on the disk. Due to the nature of reliable queue files, all the messages traversing the queue are written to disk, constantly increasing the size of the queue file.

The disk-buffer file's size should be considered as the configured `disk-buf-size()` at any point of time, even if it does not have messages in it. Truncating the disk-buffer file can slow down disk I/O operations, so syslog-ng PE does not always truncate the file when it would be possible (see the truncate-size-ratio() option). If a large disk-buffer file is not desirable, you should set the `disk-buf-size()` option to a smaller value.

CAUTION:

One Identity recommends that you do not build upon the current truncating logic of the disk-buffer files, because syslog-ng PE might pre-allocate the disk-buffer files and never truncate them in the future.

NOTE: The disk-buffer file's size does not strictly correlate to the number of stored messages. If you want to get information about the disk-buffer, use `dqtool` (for more information, see Getting the status information of disk-buffer files).

NOTE: If a queue file becomes corrupt, syslog-ng PE starts a new one. This might lead to the queue files consuming more space in total than their maximal configured size and the number of configured queue files multiplied together.

Filters

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

- Using filters describes how to configure and use filters.
- Combining filters with boolean operators shows how to create complex filters using boolean operators.
- Comparing macro values in filters explains how to evaluate macros in filters.
- Using wildcards, special characters, and regular expressions in filters provides tips on using regular expressions.
- Tagging messages explains how to tag messages and how to filter on the tags.
- 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>"); };
```

Then use the filter in a log path, for example:

```
log {
    source(s1);
    filter(<identifier>);
    destination(d1); };
```

You can also define the filter inline. For details, see Defining configuration objects inline.

Example: 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")) };
log {
    source(s1);
    filter(demo_filter);
    destination(d1); };
```

The following example does the same, but defines the filter inline.

```
log {
    source(s1);
    filter { host("example") and match("deny" value("MESSAGE")) };
    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 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 match().

TIP: Filters are often used together with log path flags. For details, see Log path flags.

Comparing macro values in filters

Starting with syslog-ng PE version 3.24 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 Customizing message format using macros and templates.

```
filter <filter-id>
       {"<macro-or-template>" operator "<value-or-macro-or-template>"};
```

Example: 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"};
```

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 13: 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 PCRE regular expressions.

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: 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 Using wildcards, special characters, and regular expressions in filters.

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 Using pattern databases, for details on creating tags in the pattern database, see The syslog-ng pattern database format.
- Tags can be also added and deleted using rewrite rules. For details, see Adding and deleting tags.

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.

- 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: Adding tags and filtering messages with tags.

Filter functions

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

Table 14: 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.
in-list()	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() and netmask6()	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 15: 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. For example:

```
filter demo_filter { host("example") and match("deny" value("MESSAGE")) };
filter inverted_demo_filter { not filter(demo_filter) }
```

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") };
```

in-list()

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 3.5 and later.

Example: 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 Templates and macros, Parsing messages with comma-separated and similar values, and 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 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 netmask().

The `netmask6()` filter is available in syslog-ng PE 5.0.8 and 5.2.23.7 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: 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: 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); };
```

Global options of syslog-ng PE

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: 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 Global options. For important global options and recommendations on their use, see Best practices and examples.

Global options

The following options can be specified in the options statement, as described in 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	
			no	client-hostname-resolved-on-the-relay	client-hostname-resolved-on-the-relay / relay-hostname-	client-hostname-resolved-on-the-relay	

	chain-hostnames() setting on the server			
	yes		no	
	keep-hostname() setting on the server		keep-hostname() setting on the server	
	yes	no	yes	no
		resolved-on-the-server		

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 and sockets.

custom-domain()

NOTE: This global option works only if the `use-fqdn()` global option is set to `yes`.

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 fully qualified domain name (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:	-1

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`).

Starting with version 7.0.93.16, the default value of this option is -1, so syslog-ng PE does not change the ownership, unless explicitly configured to do so.

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
Default:	3600

Description: Number of seconds while a successful lookup is cached.

dns-cache-expire-failed()

Accepted values:	number
Default:	60

Description: Number of seconds while a failed lookup is cached.

dns-cache-hosts()

Accepted values:	filename
Default:	unset

Description: Name of a file in `/etc/hosts` format that contains static IP->hostname mappings. Use this option to resolve hostnames locally without using a DNS. Note that any change to this file triggers a reload in syslog-ng and is instantaneous.

dns-cache-size()

Accepted values:	number 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
Default:	100

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. Increasing this number increases throughput as more messages are sent in a single batch, but also increases message latency.

The syslog-ng PE application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload syslog-ng PE or in case of network sources, the connection with the client is closed, syslog-ng PE automatically sends the unsent messages to the destination.

flush-timeout()

Accepted values:	time in milliseconds
Default:	10000

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For more information, see the `flush-lines()` option.

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng PE 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: The syslog-ng PE application can add the fractions to non-ISO8601 timestamps as well.

> NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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.

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng PE configuration file.

For example:

```
jvm-options("-Xss1M -XX:+TraceClassLoading")
```

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 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:	65536

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximum value that you can set is 268435456 bytes (256 MiB).

For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64 KiB.

NOTE: In most cases, you do not need to set `log-msg-size()` higher than 10 MiB.

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

You can use human-readable units when setting configuration options. For details, see Notes about the configuration syntax.

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 Journal files.

Example: 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: 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
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.xsyslog-ng PE 3.3 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. Note that setting the global `mark-mode()` to global causes a syntax error in syslog-ng PE.

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 LTSsyslog-ng PE 3.4 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. If set, syslog-ng changes the owner of accessed files (for example, `/dev/null`) to this value, and the permissions to the value set in the `perm()` option.

Starting with version 7.0.93.16, the default value of this option is -1, so syslog-ng PE does not change the ownership, unless explicitly configured to do so.

pass-unix-credentials()

Accepted values:	yes|no
Default:	yes

Description: Enable syslog-ng PE to collect UNIX credential information (that is, the PID, user ID, and group of the sender process) for messages received using UNIX domain sockets. Available only in syslog-ng Premium Edition 5 F5syslog-ng Premium Edition 3.7 and later. Note that collecting UNIX credential information from sockets in high-traffic environments can be resource intensive, therefore `pass-unix-credentials()` can be disabled globally, or separately for each source.

perm()

Accepted values:	permission value
Default:	-1

Description: The default permission for output files. If set, syslog-ng changes the permissions of accessed files (for example, `/dev/null`) to this value, and the onwer to the value set in the `owner()` option.

Starting with version 7.0.93.16, the default value of this option is -1, so syslog-ng PE does not change the permissions, unless explicitly configured to do so.

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 Timezones and daylight saving and 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.

reset-license-counter()

Accepted values:	yes|no
Default:	no

Description: When set to `yes`, syslog-ng PE resets the host-limit counter and the list of known hosts every day at midnight (local time). That way, you can make syslog-ng PE forget old clients that do not exist anymore. This is especially useful in large datacenters or cloud environments where the client hosts are deployed and removed frequently. Available in syslog-ng PE version 7.0.11 and later.

CAUTION:

The license-counter reset is hard-coded to midnight. If syslog-ng PE is reloaded at midnight, just before the license-counter is reset, the license-counter reset will be rescheduled for the next midnight. If this happens repeatedly, the license-counter will never be reset.

When the counter is reset, the following message is logged into the `internal()` source: `Resetting license host-counter; currently used hosts='1'`

```
options
 {
    reset-license-counter(yes);
 }
```

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 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
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.

Note that these messages are sent from the `internal()` source of syslog-ng PE and are unstructured. If you want to select which counters you want to monitor, and format the messages, use the `monitoring()` source. For details on how you can monitor syslog-ng PE statistics and metrics, see Monitoring statistics and metrics of syslog-ng.

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 Monitoring statistics and metrics of syslog-ng.

stats-max-dynamics()

Accepted values:	number
Default:	`N/A`

Description: To avoid performance issues or even overloading syslog-ng PE (for example, if a script starts to send logs from different IP addresses to syslog-ng PE), you might want to limit the number of registered dynamic counters in the message statistics. For details on message statistics, see Monitoring statistics and metrics of syslog-ng.

- **Unlimited dynamic counters:**

  If you do not use this option, dynamic counters will not be limited. This can be useful in cases where you are extremely interested in dynamic counters, and use these statistics extensively.

  CAUTION:

  In some cases, there might be even millions of dynamic counters

- **Limited dynamic counter clusters**

  To limit dynamic counters, enter a number, and only a maximum of <number> counters will be registered in the statistics.

  In practice, this means dynamic counter clusters. A program name produces one dynamic counter cluster, that can include several counters, such as `processed`, `stamp`, and so on.

  Example: Limiting dynamic counter clusters 1

  If you set `stats-max-dynamics()` to `1`, and 2 programs send messages, only one of these programs will be tracked in the dynamic counters, but it will have more than one counters.

  Example: Limiting dynamic counter clusters 2

  If you have 500 clients, and set `stats-max-dynamics()` to `1000`, you will have enough number of counters reserved for these clients, but at the same time, you limit the use of your resources and therefore protect your system from being overloaded.

- **No dynamic counters**

  To disable dynamic counters completely, set the value of this option to `0`. This is the recommended value if you do not use statistics, or if you are not interested in dynamic counters in particular (for example, the number of logs arriving from programs).

NOTE: If you set a lower value to `stats-max-dynamics()` (or, any limiting value, if this option has not been configured before) and restart syslog-ng PE, the changes will only be applied after `stats-freq()` time has passed. That is, the previously allocated dynamic clusters will only be removed after this time.

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 F1syslog-ng Premium Edition 3.3 and later. See 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
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 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.

trim-large-messages()

Accepted values:	yes|no
Default:	no

Description: Determines what syslog-ng PE does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, syslog-ng PE drops the incoming log message.
- If set to `yes`, syslog-ng PE trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter  f_trimmed  {
        tags("trimmed");
  };
  ```

  If syslog-ng PE trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in syslog-ng PE version 7.0.143.21 and later.

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 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 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: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

TIP: Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.

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: When the `use-rcptid` global option is set to `yes`, syslog-ng PE automatically assigns a unique reception ID to every received message. You can access this ID and use it in templates via the `${RCPTID}` macro. The reception ID is a monotonously increasing 48-bit integer number, that can never be zero (if the counter overflows, it restarts with 1).

This option is deprecated, use the `use-uniqid()` option instead.

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 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 RCPTID

11

TLS-encrypted message transfer

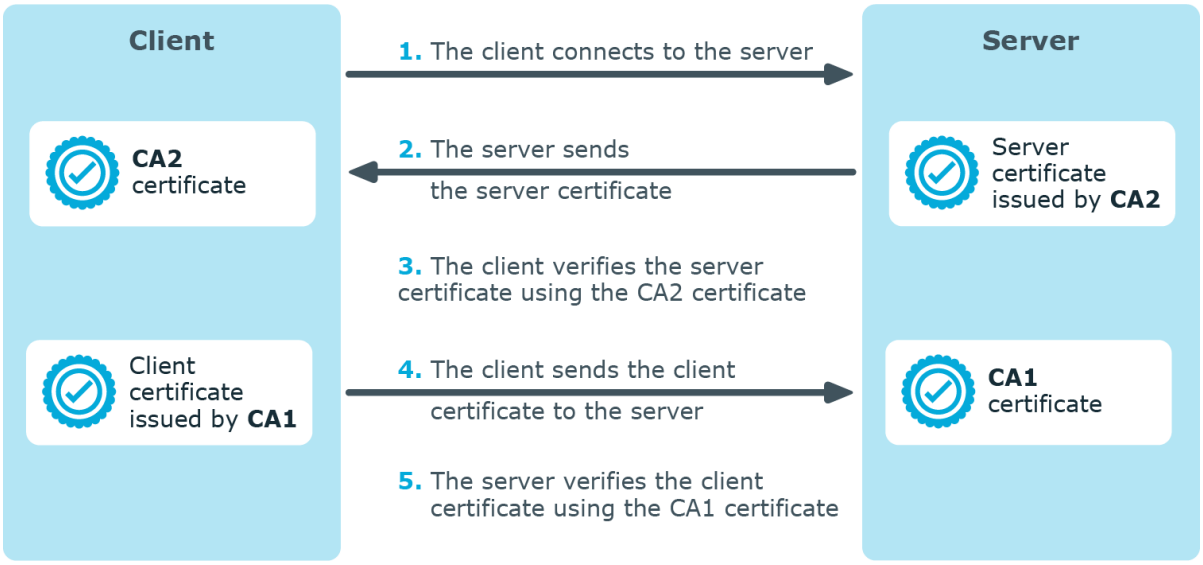
Secure logging using TLS

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.

NOTE: This chapter describes how to use TLS encryption when using the standard syslog protocols, that is, the `network()` and `syslog()` drivers, for example, to forward log messages between two syslog-ng nodes, or to send log data to syslog-ng Store Box or another log server. Other destinations that support TLS-encryption are not discussed in this chapter (for example, `http()`).

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

Figure 36: 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 more information about configuring TLS communication in syslog-ng, see Encrypting log messages with TLS.

For more information about TLS-related error messages, see Error messages.

Supported OpenSSL versions

The following list contains information about the supported OpenSSL versions in each syslog-ng PE application version.

Linux glibc 2.11

syslog-ng PE version	supported Open SSL version
7.0.1	OpenSSL 1.0.2j
7.0.2	OpenSSL 1.0.2j
7.0.3	OpenSSL 1.0.2j
7.0.4	OpenSSL 1.0.2j
7.0.5	OpenSSL 1.0.2j
7.0.6	OpenSSL 1.0.2m
7.0.7	OpenSSL 1.0.2m
7.0.8	OpenSSL 1.0.2m
7.0.9	OpenSSL 1.0.2o
7.0.10	OpenSSL 1.0.2o
7.0.11	OpenSSL 1.0.2p
7.0.12	OpenSSL 1.0.2q
7.0.13	OpenSSL 1.0.2q
7.0.14	OpenSSL 1.0.2r
7.0.15	OpenSSL 1.0.2s
7.0.16	OpenSSL 1.0.2s
7.0.17	OpenSSL 1.0.2t
7.0.18	OpenSSL 1.0.2t
7.0.19	OpenSSL 1.1.1d
7.0.20	OpenSSL 1.1.1g
7.0.21	OpenSSL 1.1.1g
7.0.22	OpenSSL 1.1.1g
7.0.23	OpenSSL 1.1.1h
7.0.24	OpenSSL 1.1.1j
7.0.25.	OpenSSL 1.1.1k
7.0.26	OpenSSL 1.1.1k
7.0.27	OpenSSL 1.1.1l

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 Secure logging using TLS.

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):

To configure TLS on the syslog-ng clients

1. **Copy the CA certificate to the client host**

   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.

2. **Issue commands**

   Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem` The result is a hash (for example, `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

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

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

3. **Add a destination statement**

   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: 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"))
       );
 };
```

4. **Include the destination in a log statement**

   Include the destination created in the Step Add a destination statement 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.

Configuring TLS on the syslog-ng server

Complete the following steps on the syslog-ng server:

To configure TLS on the syslog-ng server

1. 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.

2. 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.
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. If you want to use a password-protected key, see Password-protected keys.
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.

Example: 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")
        )
    );
};
```

5. Disable mutual authentication for the source by setting the following TLS option in the source statement: `tls( peer-verify(optional-untrusted);`

   If you want to authenticate the clients, you have to configure mutual authentication. For details, see Mutual authentication using TLS.

   For the details of the available `tls()` options, see TLS options.

Example: 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 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 Secure logging using TLS.

Configuring TLS on the syslog-ng clients

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):

To configure TLS on the syslog-ng clients

1. **Create an X.509 certificate**

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

2. **Copy the certificate and private key**

   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. If you want to use a password-protected key, see Password-protected keys.

3. **Copy the CA certificate to the client host**

   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.

4. **Issue commands**

   Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem` The result is a hash (for example, `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

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

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

5. **Add a destination statement**

   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: 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")) ); };
   ```

6. **Include destination in a log statement**

Include the destination created in the Step Add a destination statement 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.

Configuring TLS on the syslog-ng server

Complete the following steps on the syslog-ng server:

To configure TLS on the syslog-ng clients

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.

   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. If you want to use a password-protected key, see Password-protected keys.
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 TLS options.

Example: 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.

Password-protected keys

Starting with syslog-ng PE version 3.147.0.7, you can use password-protected private keys in the `network()` and `syslog()` source and destination drivers.

Restrictions and limitations

- IMPORTANT: *Hazard of data loss!* If you use password-protected keys, you must provide the passphrase of the password-protected keys every time syslog-ng PE is restarted (syslog-ng PE keeps the passphrases over reloads). The sources and

> destinations that use these keys will not work until you provide the passwords. Other parts of the syslog-ng PE configuration will be unaffected.

This means that if you use a password-protected key in a destination, and you use this destination in a log path that has multiple destinations, neither destinations will receive log messages until you provide the password. In this cases, always use the disk-buffer option to avoid data loss.

- The path and the filename of the private key cannot contain whitespaces.
- Depending on your platform, the number of passwords syslog-ng PE can use at the same time might be limited (for example, on Ubuntu 16.04 you can store 16 passwords if you are running syslog-ng PE as a non-root user). If you use lots of password-protected private keys in your syslog-ng PE configuration, increase this limit using the following command: `sudo ulimit -l unlimited`

Providing the passwords

The `syslog-ng-ctl credentials status` command allows you to query the status of the private keys that syslog-ng PE uses in the `network()` and `syslog()` drivers. The command returns the list of private keys used, and their status. For example:

```
syslog-ng-ctl credentials status
Secret store status:
/home/user/ssl_test/client-1/client-encrypted.key SUCCESS
```

If the status of a key is PENDING, you must provide the passphrase for the key, otherwise syslog-ng PE cannot use it. The sources and destinations that use these keys will not work until you provide the passwords. Other parts of the syslog-ng PE configuration will be unaffected. You must provide the passphrase of the password-protected keys every time syslog-ng PE is restarted.

The following log message also notifies you of PENDING passphrases:

```
Waiting for password; keyfile='private.key'
```

You can add the passphrase to a password-protected private key file using the following command. syslog-ng PE will display a prompt for you to enter the passphrase. We recommend that you use this method.

```
syslog-ng-ctl credentials add --id=<path-to-the-key>
```

Alternatively, you can include the passphrase in the `--secret` parameter:

```
syslog-ng-ctl credentials add --id=<path-to-the-key> --secret=<passphrase-of-the-key>
```

Or you can pipe the passphrase to the syslog-ng-ctl command, for example:

```
echo "<passphrase-of-the-key>" | syslog-ng-ctl credentials add --id=<path-to-the-key>
```

For details on the `syslog-ng-ctl credentials` command, see The syslog-ng control tool manual page.

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 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.

Available in version 3.197.0.12 and later.

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. For an example, see Configuring TLS on the syslog-ng clients. The syslog-ng PE application uses the CA certificates in this directory to validate the certificate of the peer.

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 set in the `key-file()` option. The syslog-ng PE application uses this certificate to authenticate the syslog-ng PE client on the destination server. 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:	Depends on the OpenSSL version that syslog-ng PE uses

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")
```

For a list of available algorithms, execute the `openssl ciphers -v` command. The first column of the output contains the name of the algorithms to use in the `cipher-suite()` option, the second column specifies which encryption protocol uses the algorithm (for example, `TLSv1.2`). That way, the `cipher-suite()` also determines the encryption protocol used in the connection: to disable SSLv3, use an algorithm that is available only in TLSv1.2, and that both the client and the server supports. You can also specify the encryption protocols using ssl-options().

You can also use the following command to automatically list only ciphers permitted in a specific encryption protocol, for example, `TLSv1.2`:

```
echo "cipher-suite(\"$(openssl ciphers -v | grep TLSv1.2 | awk '{print $1}' |
xargs echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

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.

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.

ecdh-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 automatically uses the highest preference curve that both peers support. If not specified, the list includes every supported curve.

Example:

```
ecdh-curve-list("prime256v1:secp384r1")
```

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`.

key-file()

Accepted values:	Filename
Default:	none

Description: Path and name of a file that contains a private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng PE application uses this private key and the

matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng PE client on the destination server.

Starting with syslog-ng PE version 3.147.0.7, you can use password-protected private keys in the `network()` and `syslog()` source and destination drivers. The path and the filename cannot contain whitespaces. For details, see Password-protected keys.

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).
- 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.

ssl-options()

Accepted values:	comma-separated list of the following options: no-sslv2, no-sslv3, no-tlsv1, no-tlsv11, no-tlsv12, none
Default:	no-sslv2

Description: Sets the specified options of the SSL/TLS protocols. Currently, you can use it to disable specific protocol versions. Note that disabling a newer protocol version (for example, TLSv1.1) does not automatically disable older versions of the same protocol (for example, TLSv1.0). For example, use the following option to permit using only TLSv1.1 or newer:

```
ssl-options(no-sslv2, no-sslv3, no-tlsv1)
```

Using `ssl-options(none)` means that syslog-ng PE does not specify any restrictions on the protocol used. However, in this case, the underlying OpenSSL library can restrict the available protocols, for example, certain OpenSSL versions automatically disable SSLv2.

This option is available in syslog-ng PE3.77.0 and newer.

Example: Using `ssl-options`

The following destination explicitly disables SSL and TLSv1.0

```
destination demo_tls_destination {
       network("172.16.177.147" port(6514)
       transport("tls")
       tls( ca-dir("/etc/syslog-ng/ca.d")
              key-file("/etc/syslog-ng/cert.d/clientkey.pem")
              cert-file("/etc/syslog-ng/cert.d/clientcert.pem")
              ssl-options(no-sslv2, no-sslv3, no-tlsv1) )
       ); };
```

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, note the following:

- First, the `trusted-keys()` parameter is checked. If the fingerprint of the peer is listed, the certificate validation is performed.
- If the fingerprint of the peer is not listed in the `trusted-keys()` parameter, the `trusted-dn()` parameter is checked. If the DN of the peer is not listed in the `trusted-dn()` parameter, the authentication of the peer fails and the connection is closed.

Advanced Log Transfer Protocol

Logging using ALTP

CAUTION:

Note that in earlier versions of syslog-ng Premium Edition, Advanced Log Transfer Protocol is called Reliable Log Transfer Protocol (RLTP).

The syslog-ng PE application can send and receive log messages in a reliable way over the TCP transport layer using the Advanced Log Transfer Protocol (ALTP). ALTP 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 mechanisms of syslog-ng PE's flow-control and the reliable disk-buffer option, thus providing the best way to prevent message loss. The sender detects which messages the receiver has 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).

ALTP also allows for encrypted and non-encrypted connections to be received on the same port, using a single source driver.

NOTE: Because of the communication overhead, the ALTP 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 Premium Edition 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 ALTP:

- **If you use ALTP together with the non-reliable disk-buffer option, it is possible to lose logs.**
- **When sending logs through a relay that is using the non-reliable disk-buffer option, it is possible to lose logs if the relay crashes.**
- **When sending logs through a relay that is using the non-reliable disk-buffer option, 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 ALTP protocol works on top of TCP, and can use STARTTLS for encryption. ALTP supports IPv4 and IPv6 addresses. Inside the ALTP message, the message can use any format, for example, RFC3164 (BSD-syslog) or RFC5424 (IETF-syslog). The default port of ALTP is `35514`.

ALTP can be added to the configuration like a transport protocol within the `syslog()` driver and the `network()` driver.

How ALTP connections work

This procedure summarizes how two syslog-ng PE hosts (a sender and a receiver) communicate using the Advanced Log Transfer Protocol (ALTP).

Prerequisites

The sender (also called the client) is the host that has ALTP configured in its destination driver. The receiver (also called the server) is the host that has ALTP configured in its source driver.

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 ALTP, 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 `batch-size()` parameter of the sender.

For optimal performance when sending messages to a syslog-ng PE server, make sure that the `batch-size()` 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 acknowledgment 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* ALTP, dropped because of filter settings, or written to the disk-buffer. (If the messages are forwarded using ALTP, see Using ALTP in a client-relay-server scenario.)

7. After receiving the acknowledgment, the sender sends another batch of messages.

Using ALTP in a client-relay-server scenario

CAUTION:

Note that in earlier versions of syslog-ng Premium Edition, Advanced Log Transfer Protocol is called Reliable Log Transfer Protocol (RLTP).

You can use ALTP between multiple syslog-ng PE hosts, for example, in a client-relay-server scenario. In this case, the communication described in How ALTP 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 acknowledgment 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-acknowledgment-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 for the server to acknowledge the messages.

ALTP options

The following options are specific to the ALTP protocol. Note that when using ALTP in a source or a destination, the options of the `syslog()` or the `network()` driver can be used as well.

CAUTION:

Note that in earlier versions of syslog-ng Premium Edition, Advanced Log Transfer Protocol is called Reliable Log Transfer Protocol (RLTP).

allow-plain-compress()

Accepted values:	yes | no
Default:	no

Description: Enable on-the-wire compression in the ALTP 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-plain-compress()` option can be used in source and destination drivers as well. Available in syslog-ng PE 7.0.12 and later. Note that in earlier versions, the name of this option was `allow-compress()`.

batch-size()

Accepted values:	number
Default:	1000

Description: Specifies the number of lines that 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 network, but also increases message latency. Available in syslog-ng PE 7.0.10 and later.

message-acknowledgment-timeout()

Type:	number (seconds)
Default:	900

Description: When the receiver (syslog-ng PE server) receives and successfully processes a message, it sends an acknowledgment 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 acknowledgment, which is governed by the `message-acknowledgment-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 the server side and `tls-required(no)` on the 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 ALTP 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 ALTP

CAUTION:

Note that in earlier versions of syslog-ng Premium Edition, Advanced Log Transfer Protocol is called Reliable Log Transfer Protocol (RLTP).

Example: Simple ALTP connection

The sender and the receiver use ALTP 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_altp  {
            network(
                  ip("127.0.0.1")
                  port("5555")
                  transport(altp)
                  ip-protocol(4)
            );
 };
```

Sender configuration (syslog-ng PE client):

```
destination  d_network_altp  {
            network(
                  "127.0.0.1"
                  port("5555")
                  transport(altp)
                  ip-protocol(4)
            );
 };
```

Example: ALTP with TLS encryption

The following example configures a sender and a receiver to communicate using ALTP. 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_altp {
            syslog(
                  ip("127.0.0.1")
                  port("4444")
                  transport(altp(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_altp {
            syslog(
                  "127.0.0.1"
                  port("4444")
                  transport(altp(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")
                  )
            );
};
```

Reliability and minimizing the loss of log messages

Introduction

Advanced Log Transfer Protocol (ALTP) interacts with flow control and the disk-buffer option 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, the disk-buffer option, and ALTP 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

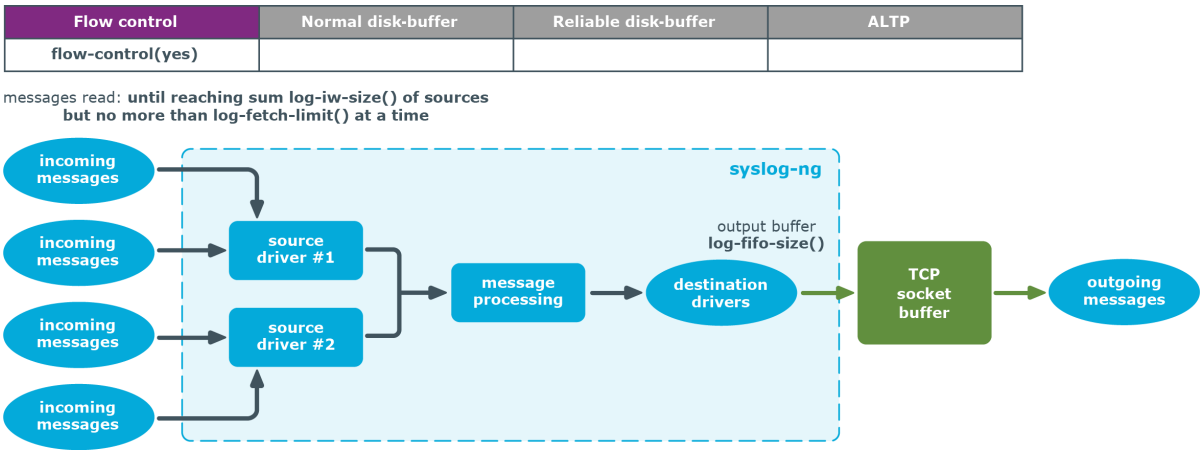
Any of the mechanisms that syslog-ng PE uses to prevent or minimize the loss of log messages only works if the hardware and operating system work normally. When there is an issue with the hardware or operating system that the application and syslog-ng PE run on, log loss may occur. Issues include operating system crash (for example, kernel panic), memory errors, disk errors, power outage, and so on.

Flow control, no disk-buffer option, no ALTP

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 37: Flow control, no disk-buffer option, no ALTP



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*: With 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 acknowledgment 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 acknowledgment 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 the disk-buffer option. For details, see Flow control, normal disk-buffer option, no ALTP on page 758 and Flow control, reliable disk-buffer option, no ALTP on page 760.
- *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 the disk-buffer option. For details, see Flow control, normal disk-buffer option, no ALTP on page 758 and Flow control, reliable disk-buffer option, no ALTP on page 760.
- *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.

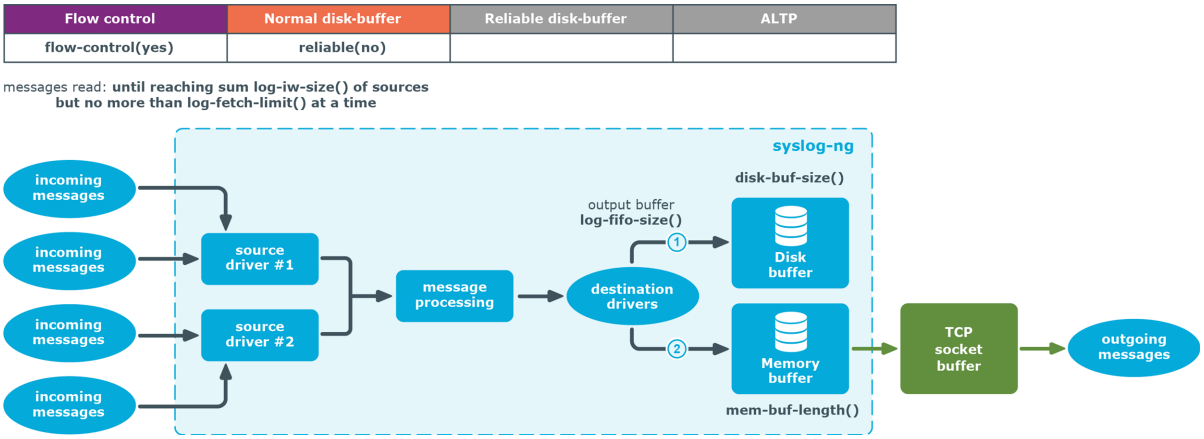
Flow control, normal disk-buffer option, no ALTP

How it works

1. syslog-ng PE puts messages into the disk-buffer file (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 file 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 38: Flow control, normal disk buffering, no ALTP



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 the disk-buffer option. For details, see Example: Example configuration of the normal disk-buffer option on page 759.

Example: Example configuration of the normal disk-buffer option

```
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 file 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 the disk-buffer option is not 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 file 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 file 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 file.

Drawbacks

One drawback of using the disk-buffer option 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*: With 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 acknowledgment 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 acknowledgment 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-buffer option, no ALTP

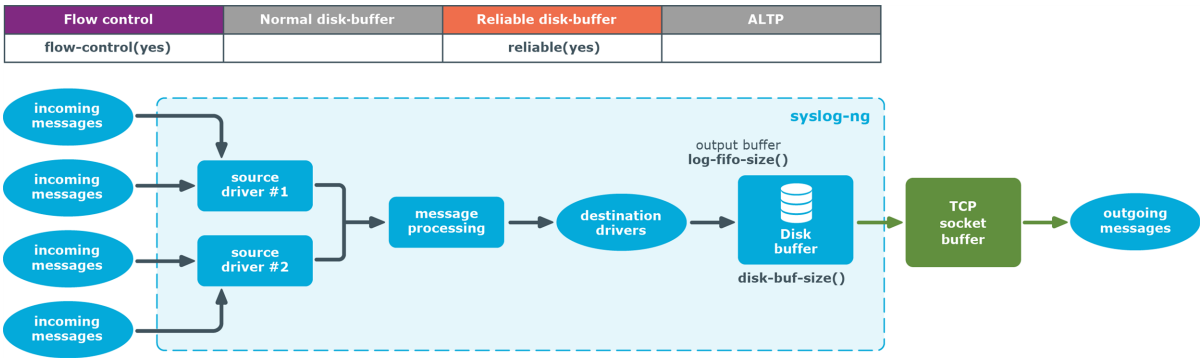
How it works

syslog-ng puts messages into the disk-buffer file, until the disk-buffer file 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 39: Flow control, reliable disk-buffer, no ALTP



How to set key parameters

Set `flags(flow-control)` in the log path.

Configure the disk-buffer option. For details, see Example: Example configuration of the reliable disk-buffer on page 761.

Example: Example configuration of the reliable disk-buffer

```
disk-buffer(
        mem-buf-size(10485760) # storing 10 MB 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 the disk-buffer option, those messages are considered delivered that have been written to the disk-buffer file. As soon as the disk-buffer file 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 the disk-buffer option is not 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 file has been filled to its full capacity that the sources are stopped. This allows 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 file 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 file, which, again, results in more reliability.

  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 file.

- *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 option is persistent and when the disk-buffer file 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 file and the output buffer. After the restart, syslog-ng PE sends the saved messages to the destination.

Drawbacks

One drawback of using the reliable disk-buffer option 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 the normal disk-buffer option.

This configuration does not provide protection against the loss of log messages in the following situations:

- *TCP error*: With 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 acknowledgment 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 acknowledgment 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-buffer option, ALTP

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 `batch-size()`).
2. The relay writes messages to the disk-buffer file.
3. Once messages have been written to the disk-buffer file, the relay returns an acknowledgment to the client.
4. The relay sends messages to the server in batches (set via `batch-size()`).
5. When the server has successfully received and processed the messages in the batch, it sends an acknowledgment of the processed messages to the relay.

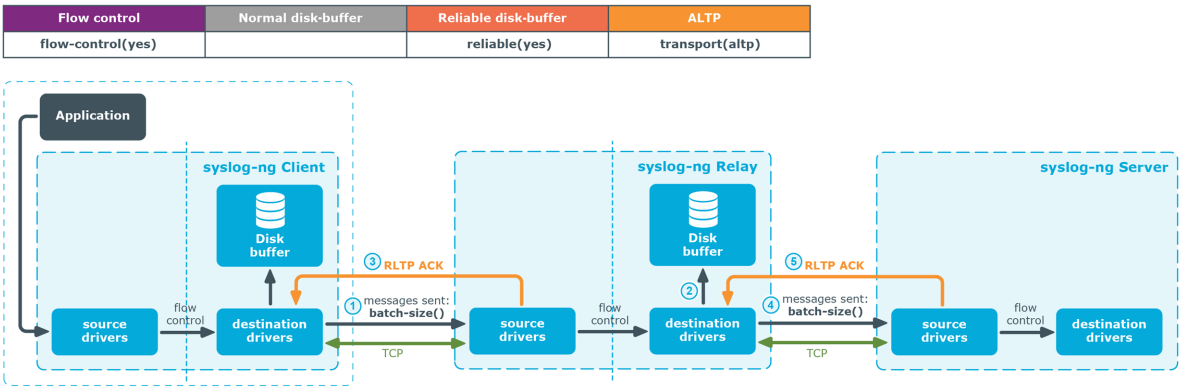
   It is only at this point that the relay removes log messages from the disk-buffer file because this is when logs are considered "delivered" to the server.

   After receiving the acknowledgment, the sender sends another batch of messages.

This configuration gives you the greatest degree of protection against log message loss. ALTP provides acknowledgment 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 delivered only when the syslog-ng PE application has received an acknowledgment 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-buffer.

Figure 40: Flow control, reliable disk-buffer, ALTP



How to set key parameters

Set `flags(flow-control)` in the log path.

Configure the disk-buffer option. For details, see Example: Example configuration of the reliable disk-buffer on page 764.

Example: Example configuration of the reliable disk-buffer

```
disk-buffer(
        mem-buf-size(10485760) # storing 10 MB messages in memory and
on disk
        disk-buf-size(2147483648) # storing 2 GB of messages only on
disk
        reliable(yes)
 )
```

Enable ALTP by setting `transport(rltp)`. For details, see ALTP options on page 751.

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 reliable disk buffering and ALTP, 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*: With 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 acknowledgment 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, ALTP introduces flow control at the application layer. This means that log messages are only considered delivered when the ALTP 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 file do not get lost.
- *Message loss outside of syslog-ng PE*: One of the advantages of this configuration over when the disk-buffer option is not 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 file has been filled to its full capacity that the sources are stopped. This allows 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 file 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 file, which, again, results in more reliability.

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 file.

- *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 option is persistent and when the disk-buffer file 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 file 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 straightforward 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.

12

Manipulating messages

This chapter explains the methods that you can use to customize, reformat, and modify log messages using syslog-ng Premium Edition.

- Customizing message format using macros and templates explains how to use templates and macros to change the format of log messages, or the names of logfiles and database tables.
- Modifying messages using rewrite rules describes how to use rewrite rules to search and replace certain parts of the message content.
- 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 using macros and templates

The following sections describe how to customize the names of logfiles, and also how to use templates, macros, and template functions.

- Formatting messages, filenames, directories, and tablenames explains how macros work.
- Modifying messages using rewrite rules describes how to use macros and templates to format log messages or change the names of logfiles and database tables.
- Macros of syslog-ng PE lists the different types of macros available in syslog-ng PE.
- Using template functions explains what template functions are and how to use them.
- 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 Templates and macros.
- For a list and description of the macros available in syslog-ng PE, see Macros of syslog-ng PE.
- For details on using custom macros created with CSV parsers and pattern databases, see parser: Parse and segment structured messages and 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 Macros of syslog-ng PE. 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.

If you do not want to enable the `template-escape()` option (which is rarely needed), you can define the template without the enclosing braces.

```
template <template-name> "<template-expression>";
```

You can also refer to an existing template from within a template. The result of the referred template will be pasted into the second template.

```
template first-template "sample-text";
template second-template "The result of the first-template is: $(template first-template)";
```

If you want to use a template only once, you can define the template inline, for example:

```
destination d_file {
      file ("/var/log/messages" template("${ISODATE} ${HOST} ${MSG}\n") );
 };
```

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.

Macro names are case-sensitive, that is, `"$message"` and `"$MESSAGE"` are not the same.

To use a literal `$` character in a template, you have to escape it. In syslog-ng PE versions 4.0-4.23.4 and earlier, use a backslash (`\$`). In version 5.03.5 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}
```

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.)

Example: 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"); };
destination d_file {
    file("/var/log/messages" template(t_demo_filetemplate)); };
```

If you do not want to enable the `template-escape()` option (which is rarely needed), you can define the template without the enclosing braces. The following two templates are equivalent.

```
template t_demo_template-with-braces {
    template("${ISODATE} ${HOST} ${MSG}\n");
};
template t_demo_template-without-braces "${ISODATE} ${HOST} ${MSG}\n";
```

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")
);
 };
```

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 three 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.
- `C_` prefix, for example, `${C_DATE}`: `${C_DATE}` is the current date, that is when syslog-ng PE processes the message and resolves the macro. 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 `${C_USEC}` in a filter and in a destination filename, their values will be different even for the same 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`
- `MSEC`
- `SEC`
- `STAMP`
- `TZ`
- `TZOFFSET`
- `UNIXTIME`
- `USEC`
- `WEEK`
- `WEEK_DAY`
- `WEEK_DAY_ABBREV`
- `WEEK_DAY_NAME`

- YEAR
- YEAR_DAY

Hard versus 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 3.2syslog-ng PE 3.4 and later.

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 parser: Parse and segment structured messages and Using parser results in filters and templates, respectively.

DATE, C_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, C_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_NAME

Description: Name of the log file (including its path) from where syslog-ng PE received the message (only available if syslog-ng PE received the message from a file or a wildcard-file source). If you need only the path or the filename, use the `dirname` and `basename` template functions.

FULLDATE, C_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 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 Using name resolution in syslog-ng.

HOUR, C_HOUR, R_HOUR, S_HOUR

Description: The hour of day the message was sent.

HOUR12, C_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 3.2syslog-ng PE 3.4 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 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 Using name resolution in syslog-ng.

ISODATE, C_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 PE can produce fractions of a second (for example, milliseconds) in the timestamp by using the `frac-digits()` global or per-destination option.

NOTE: As syslog-ng PE is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, syslog-ng PE will truncate the fraction seconds in the timestamps after 6 digits.

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 PRIORITY or LEVEL for details.

LOGHOST

Description: The hostname of the computer running syslog-ng PE — it returns the same result as the `hostname` command.

MESSAGE

Description: Text contents of the log message without the program name and pid. The program name and the pid together are available in the `${MSGHDR}` macro, and separately in the ${PROGRAM} and ${PID} 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)`.

The ${MSG} macro is an alias of the ${MESSAGE} macro: using ${MSG} in syslog-ng PE is equivalent to ${MESSAGE}.

Note that before syslog-ng version 3.0, the ${MESSAGE} macro included the program name and the pid. In syslog-ng 3.0, the `${MESSAGE}` macro became equivalent with the `${MSGONLY}` macro.

MIN, C_MIN, R_MIN, S_MIN

Description: The minute the message was sent.

MONTH, C_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, C_MONTH_ABBREV, R_MONTH_ABBREV, S_MONTH_ABBREV

Description: The English abbreviation of the month name (3 letters).

MONTH_NAME, C_MONTH_NAME, R_MONTH_NAME, S_MONTH_NAME

Description: The English name of the month name.

MONTH_WEEK, C_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.

MSEC, C_MSEC, R_MSEC, S_MSEC

Description: The millisecond the message was sent.

Available in syslog-ng PE version 4 F23.4 and later.

MSG

The ${MSG} macro is an alias of the ${MESSAGE} macro, using ${MSG} in syslog-ng PE is equivalent to ${MESSAGE}. For details on this macro, see MESSAGE.

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. Starting with syslog-ng PE 3.0, the following macros are equivalent: ${MSGONLY}, ${MSG}, ${MESSAGE}. For consistency, use the ${MESSAGE} macro. For details, see MESSAGE.

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 PRIORITY or LEVEL for details.

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.

RAWMSG

Description: The original message as received from the client. Note that this macro is available only in 7.0.93.16 and later, and only if syslog-ng received the message using the `default-network-drivers-ng()` source, or the source receiving the message has the `store-raw-message` flag set.

RCPTID

Description: When the `use-rcptid` global option is set to `yes`, syslog-ng PE automatically assigns a unique reception ID to every received message. You can access this ID and use it in templates via the `${RCPTID}` macro. The reception ID is a monotonously increasing 48-bit integer number, that can never be zero (if the counter overflows, it restarts with 1).

RUNID

Description: An ID that changes its value every time syslog-ng PE is restarted, but not when reloaded.

SDATA, .SDATA.SDID.SDNAME

Description: The syslog-ng application automatically parses the STRUCTURED-DATA part of IETF-syslog messages, which can be referenced in macros. The `${SDATA}` macro references the entire STRUCTURED-DATA part of the message, while structured data elements can be referenced using the `${.SDATA.SDID.SDNAME}` macro.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: 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, C_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. It not lost when syslog-ng PE is reloaded, but it is reset when syslog-ng PE is 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.

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 F13.4 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, C_TZ, R_TZ, S_TZ

Description: An alias of the ${TZOFFSET} macro.

TZOFFSET, C_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, C_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 use-uniqid() and RCPTID.

Available in syslog-ng PE version 5 F23.7 and later.

USEC, C_USEC, R_USEC, S_USEC

Description: The microsecond the message was sent.

Available in syslog-ng PE version 4 F23.4 and later.

YEAR, C_YEAR, R_YEAR, S_YEAR

Description: The year the message was sent.

WEEK, C_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_DAY_ABBREV, C_WEEK_DAY_ABBREV, R_WEEK_DAY_ABBREV, S_WEEK_DAY_ABBREV

Description: The 3-letter English abbreviation of the name of the day the message was sent, for example, `Thu`.

WEEK_DAY, C_WEEK_DAY, R_WEEK_DAY, S_WEEK_DAY

Description: The day of the week as a numerical value (1-7).

WEEKDAY, C_WEEKDAY, R_WEEKDAY, S_WEEKDAY

Description: These macros are deprecated, use `${WEEK_DAY_ABBREV}`, `${R_WEEK_DAY_ABBREV}`, `${S_WEEK_DAY_ABBREV}` instead. The 3-letter name of the day of week the message was sent, for example, `Thu`.

WEEK_DAY_NAME, C_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. A template function can have maximum 64 parameters. 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 the available template functions, see the descriptions of the individual template functions in Template functions of syslog-ng PE.

You can define your own template function as a regular configuration object (for example, to reuse the same function in different places in your configuration).

Declaration

```
template-function <name-of-the-template-function> "<template-expression-using-strings-macros-template-functions>";
```

Example: Using custom template functions

The following template function can be used to reformat the message. It adds the length of the message to the message template.

```
template-function my-template-function "${ISODATE} ${HOST} message-length=$(length "${MSG}") ${MSG}";
destination d_file {
       file("/tmp/mylogs.log" template("$(my-template-function)\n")); };
```

You can also refer to existing templates in your template function.

```
template my-custom-header-template "${ISODATE} ${HOST_FROM} ${MSGHDR}";
template-function my-template-function "$(my-custom-header-template) message-length=$(length "${MSG}") ${MSG}";
```

Template functions of syslog-ng PE

The following template functions are available in syslog-ng PE.

base64-encode

Syntax:

```
$(base64-encode argument)
```

Description: You can use the `base64-encode` template function to base64-encode strings and macros. The template function can receive multiple parameters (maximum 64). In this case, syslog-ng PE joins the parameters into a single string and encodes this string. For example, `$(base64-encode string1 string2)` is equivalent to `$(base64-encode string1string2)`.

Available in syslog-ng PE version 3.187.0.11 and later.

basename

Syntax:

```
$(basename argument)
```

Description: Returns the filename from an argument (for example, a macro) that contains a filename with a path. For example, `$(basename "/var/log/messages.log")` returns `messages.log`. To extract the path, use the `dirname` template function.

Available in syslog-ng PE version 3.107.0.3 and later.

dirname

Syntax:

```
$(dirname argument)
```

Description: Returns the path (without the filename) from an argument (for example, a macro) that contains a filename with a path. For example, `$(dirname "/var/log/messages.log")` returns `/var/log` path. To extract the filename, use the `basename` template function.

Available in syslog-ng PE version 3.107.0.3 and later.

echo

Syntax:

```
$(echo argument)
```

Description: Returns the value of its argument. Using `$(echo ${HOST})` is equivalent to `${HOST}`.

env

Syntax:

```
$(env <environment-variable>)
```

Description: Returns the value of the specified environment variable. Available in syslog-ng PE3.57 and later.

format-cef-extension

syslog-ng PE includes a 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 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: 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-cim

Syntax:

```
$(format-cim)
```

Description: Formats the message into Splunk Common Information Model (CIM) format. Applications that can receive messages in CIM format include Kibana, logstash, and Splunk. Applications that can be configured to log into CIM format include nflog and the Suricata IDS engine.

```
destination d_cim {
       network("192.168.1.1" template("$(format-cim)\n"));
 };
```

You can find the exact source of this template function in the syslog-ng PE GitHub repository.

NOTE: To use the `format-cim()` template function, syslog-ng PE must be compiled with JSON support. For details, see Compiling options of syslog-ng PE . To see if your syslog-ng PE binary was compiled with JSON support, execute the `syslog-ng --version` command.

format-ewmm

Syntax:

```
$(format-ewmm)
```

Description: The `format-ewmm` template function converts the message into the Enterprise-wide message model (EWMM) format. Available in version 7.0.93.16 and later.

The following is a sample log message in EWMM format.

```
<13>1 2018-05-13T13:27:50.993+00:00 my-host @syslog-ng - - -
{"MESSAGE":"<34>Oct 11 22:14:15 mymachine su: 'su root' failed for username on
/dev/pts/8","HOST_FROM":"my-host","HOST":"my-host","FILE_NAME":"/tmp/in","._
TAGS":".source.s_file"}
```

format-flat-json

Syntax:

```
$(format-flat-json parameters)
```

Description: The `format-flat-json` template function is identical to the `format-json` template function, but nested JSON objects are flattened in the output. If you have to forward your log messages in JSON format, but the receiving application cannot handle nested JSON objects, use the `format-flat-json` template function.

Example: Flattened JSON output

The following example shows the difference between nested and flattened JSON objects.

- The output of `$(format-json a.b.c=1)` is a nested JSON object (whitespace added for better readability):

```
{
     "a": {
        "b": {
          "c": "1"
        }
     }
}
```

- The output of `$(format-flat-json a.b.c=1)` is a flattened JSON object (whitespace added for better readability):

```
{
 "a.b.c":  "1"
 }
```

For details on formatting log messages into JSON format, see .

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 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 Specifying data types in value-pairs.

Example: 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 value-pairs().

Example: 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"));
 };
```

geoip2

Syntax:

```
$(geoip2 --database <path-to-geoip2-database-file>
    [ --field "registered_country.names.ru" ] ${HOST})
```

Description: This template function extracts specific fields from the mmdb database using the `--field` parameter. If you omit this parameter, it returns the 2-letter country code of any IPv4/IPv6 address or host.

NOTE: This template function is available only if syslog-ng PE has been compiled with geoip2 support. To enable it, use the `--enable-geoip` compiling option.

To retrieve additional GeoIP information, see Looking up GeoIP2 data from IP addresses.

Starting with version 3.247.0.17, syslog-ng PE tries to automatically detect the location of the database. If that is successful, the `database()` option is not mandatory.

graphite-output

Syntax:

```
$(graphite-output parameters)
```

Description: Available in syslog-ng PE3.67.0 and later (Originally appeared in the syslog-ng PE incubator for syslog-ng 3.5). This template function converts value-pairs from the incoming message to the Graphite plain text protocol format. It is ideal to use with the messages generated by the monitor-source plugin (currently available in the syslog-ng incubator project).

For details on selecting value-pairs in syslog-ng PE and for possibilities to specify which information to convert to Graphite plain text protocol format, see Structuring macros,

metadata, and other value-pairs. Note that the syntax of `graphite-output` is different from the syntax of `value-pairs()`: `graphite-output` uses a the command-line syntax used in the format-json template function.

Example: Using the graphite-output template function

The following configuration example shows, how to send value-pairs with names starting with "`vmstat.`" to Graphite running on `localhost`, port `2003`:

```
destination d_graphite {
       network( host("localhost") port(2003) template("$(graphite-output
--key vmstat.*)"));
 };
```

grep

Syntax:

```
$(grep condition value-to-select)
```

Description: The `grep` template function can search a message context when correlating messages (for example, when you use a pattern database or the grouping-by parser). The `context-lookup` template function requires a condition (a filter or a string), and returns a specific macro or template of the matching message (for example, the ${MESSAGE} field of the message).

Example: 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})
```

You can 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 email 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 md5. 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.

NOTE: These template functions are available only if syslog-ng PE has been compiled with the `--enable-ssl` compile option and the `tfhash` module has been loaded.

By default, syslog-ng PE loads every available module. For details, see Loading modules.

Example: 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: 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("MESSAGE"));};
```

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: Using pattern databases and the if template function

The following example returns `violation` if the `username` name-value pair of a message 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"))</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: 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. For details, see Loading modules.

List manipulation

The `list-*` template functions allow you to manipulate comma-separated lists. Such lists represent a simple array type in syslog-ng PE. Note the following about formatting lists:

- Values are separated by commas, for example, `"item1","item2","item3"`. The single-element list is an element without a comma.
- You can use shell-like quotation to embed commas, for example, `"item1","ite\,m2","item3"`.
- Empty values are skipped (except if they are quoted)

These template functions return a well-formed list, properly encoding and quoting all elements. If a template function returns a single element, all quotation is decoded and the value contains the literal value.

Starting with syslog-ng PE version 3.107.0.15, the following list-related template functions are available. Certain functions allow you to reference an element using its number: note that the list index starts with zero, so the index of the first element is 0, the second element is 1, and so on.

list-append

Syntax:

```
$(list-append ${list} ${name-value-pair1} ${name-value-pair2} ... )
```

Description: Returns a list and appends the values of the specified name-value pairs to the end of the list. You can also append elements to an empty list, for example, `$(list-append '' 'element-to-add')`

list-concat

Syntax:

```
$(list-concat ${name-value-pair1} ${name-value-pair2} ... )
```

The commas between the parameters are optional.

Description: This template function creates (concatenates) a list of the values it receives as parameter. The values can be single values (for example, `${HOST}`) or lists.

For example, the value of the `$(list-concat ${HOST}, ${PROGRAM}, ${PID})` is a comma-separated list.

You can concatenate existing lists into a single list using:

```
$(list-concat ${list1} ${list2})
```

list-count

Syntax:

```
$(list-count ${list} )
```

Description: Returns the number of elements in the list.

list-head

Syntax:

```
$(list-head ${list} )
```

Description: Returns the first element of the list, unquoted.

list-nth

Syntax:

```
$(list-nth <index-number> ${list} )
```

Description: Returns the nth element of the list, unquoted. Note that the list index starts with zero, so `(list-nth 1 ${list} )` returns the second element, and so on.

list-tail

Syntax:

```
$(list-tail ${list} )
```

Description: Returns the list without the first element. For example, if the `${mylist}` list contains the `one, two, three` elements, then `$(list-tail ${mylist} )` returns `two, three`.

list-slice

Syntax:

```
$(list-slice <from>:<to> ${list} )
```

Description: Returns the specified subset of the list. Note that the list index starts with zero, for example, `$(list-slice 1:2 ${list} )` returns the second and third element of the list, and so on.

You can omit the from or to index if you want to start the subset from the beginning or end of the list, for example: `3:` returns the list starting with the 4th element, while `:3` returns the first four elements.

Negative numbers select an element from the end of the list, for example, `-3:` returns the last three element of the list.

length

Syntax:

```
$(length "<macro>")
```

Description: Returns the length of the macro in characters, for example, the length of the message. For example, the following filter selects messages that are shorter than 16 characters:

```
f_short {
      match ('-', value ("$(if ($(length "${MESSAGE}") <= 16) "-" "+")"));
};
```

lowercase

Syntax:

```
$(lowercase "<macro>")
```

Description: Returns the lowercase version of the specified string or macro. For example, the following example uses the lowercase version of the hostname in a directory name:

```
destination d_file {
      file ("/var/log/${MONTH}/${DAY}/$(lowercase "${HOST}")/messages");
};
```

Available in syslog-ng PE3.57.0 and later.

Numerical operations

Syntax:

```
$(<operation> "<value1>" "<value2>")
```

Description: These template functions allow you to manipulate numbers, that is, to perform addition (+), substraction (-), multiplication (*), division (/), and modulus (%). All of them require two numeric arguments. The result is `NaN` (Not-a-Number) if the parameters are not numbers, cannot be parsed, or if a division by zero would occur. For example, to add the value of two macros, use the following template function:

```
$(+ "${<MACRO1>}" "${<MACRO2>}");
```

When you are correlating messages and a name-value pair contains numerical values in the messages, you can calculate the lowest (min), highest (max), total (sum), and mean (average) values. These calculations process every message of the correlation context. For details on message correlation, see Correlating log messages. For example, if the messages of the context have a `.myfields.load` name-value pair, you can find the highest load value using the following template function.

```
$(max ${.myfields.load})
```

or

Syntax:

```
$(or <macro1> <macro2>)
```

Description: This template function returns the first non-empty argument.

padding

Syntax:

```
$(padding <macro> <width> <prepended-character-or-string>)
```

Description: This template function returns the value of its first parameter (a string or macro), prepended with a string. This string is `<width>` long, and repeats the character or string set in the third parameter. If you use a single character, it is added `<width>` times. If you use a string, it is repeated until its length reaches `<width>`. The default padding character is ' ' (space). For example:

Example: Using the padding template function

If the value of the `${MESSAGE}` macro is `mymessage`, then the output of the `padding()` template function is the following:

```
$(padding ${MESSAGE} 10 X)
```

Output: `XXXXXXXXXXmymessage`

```
$(padding ${MESSAGE} 10 foo)
```

Output: `foofoofoofmymessage`

python

Syntax:

```
$(python <name-of-the-python-method-to-use> <arguments-of-the-method>)
```

Description: This template function enables you to write a custom template function in Python. You can define a Python block in your syslog-ng PE configuration file, define one or more Python functions in it, and use the methods as template functions. If you use a Python block, syslog-ng PE embeds a Python interpreter to process the messages.

The following points apply to using Python blocks in syslog-ng PE in general:

- Only the default Python modules are available (that is, you cannot import external Python modules, and One Identity does not support using external Python modules).
- The syslog-ng PE application uses its own Python interpreter (shipped with the default syslog-ng PE installation) instead of the system's Python interpreter.
- The syslog-ng PE application is shipped with Python version 3.8.
- The Python block must be a top-level block in the syslog-ng PE configuration file.

- If you store the Python code in a separate Python file and only include it in the syslog-ng PE configuration file, make sure that the PYTHON_PATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start syslog-ng PE manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`

  In production, when syslog-ng PE starts on boot, you must configure your startup script to include the Python path. The exact method depends on your operating system. For recent Red Hat Enterprise Linux, Fedora, and CentOS distributions that use systemd, the `systemctl` command sources the `/etc/sysconfig/syslog-ng` file before starting syslog-ng PE. (On openSUSE and SLES, `/etc/sysconfig/syslog` file.) Append the following line to the end of this file: `PYTHONPATH="<path-to-your-python-file>"`, for example, `PYTHONPATH="/opt/syslog-ng/etc"`
- The Python object is initiated every time when syslog-ng PE is started or reloaded.

  CAUTION:

  If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.
- The Python block can contain multiple Python functions.
- Using Python code in syslog-ng PE can significantly decrease the performance of syslog-ng PE, especially if the Python code is slow. In general, the features of syslog-ng PE are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The syslog-ng PE application does not do any of this.
- Python error messages are available in the `internal()` source of syslog-ng PE.
- You can access the name-value pairs of syslog-ng PE directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of syslog-ng PE. For details, see Logging from your Python code.

- **Support disclaimer**

  CAUTION:

  This is a Preview Feature, which provides an insight to planned enhancements to functionality in the product. Consider this Preview Feature a work in progress, as it may not represent the final design and functionality.

  This feature has completed QA release testing, but its full impact on production systems has not been determined yet, and potential future changes in functionality and the user interface may result in compatibility issues in your current settings.

  One Identity recommends the following:

  - **Consider the potential risks when using this functionality in a production environment.**
  - **Consider the Support Policy on Product Preview Features before using this functionality in a production environment.**
  - **Closely and regularly keep track of official One Identity announcements about potential changes in functionality and the user interface. If these potential changes affect your configuration, check the changes you have to make in your configuration, otherwise your syslog-ng PE application may not start after upgrade.**
  - **Always perform tests prior to upgrades in order to avoid the risks mentioned.**

  However, you are welcome to try this feature and if you have any feedback, Contact One Identity.

  Support Policy on Product Preview Features

  The One Identity Support Team will:

  - **Accept and review each service request opened regarding a Preview Feature.**
  - **Consider all service requests relating to a Preview Features as severity level 3.**
  - **Provide best effort support to resolve any issues relating to a Preview Feature.**
  - **Work with customers to log any product defects or enhancements relating to Preview Features.**
  - **Not accept requests for escalations regarding Preview Features.**
  - **Not provide after-hours support for Preview Features.**

  Using Python in syslog-ng PE is recommended only if you are familiar with both Python and syslog-ng PE. One Identity is not responsible for the quality, resource requirements, or any bugs in the Python code, nor any syslog-ng PE crashes,

message losses, or any other damage caused by the improper use of this feature, unless explicitly stated in a contract with One Identity.

The following points apply to Python parsers.

- The first argument in the definition of the Python function is the actual log message. This is implicitly passed to the function, you do not have to use it in the template function.
- The value of the template function is return value of the Python function.
- To reference a name-value pair or a macro in the Python code, use the following format. For example, if the first argument in the definition of the function is called `log-message`, the value of the HOST macro is `log-message['HOST']`, and so on. (The `log-message` contains the entire log message (not just the text body) in a structure similar to a Python dict, but it is actually an object.)
- You can define new name-value pairs in the Python function. For example, if the first argument in the definition of the function is called `log-message`, you can create a new name-value pair like this: `log_message["new-macro-name"]="value"`. This is useful when you parse a part of the message from Python, or lookup a value based on data extracted from the log message.

  Note that the names of the name-value pairs are case-sensitive. If you create a new name-value pair called `new-macro-name` in Python, and want to reference it in another part of the syslog-ng PE configuration file (for example, in a template), use the `${new-macro-name}` macro.
- You cannot override hard macros (see Hard versus soft macros).
- To list all available keys (names of name-value pairs), use the `log_message.keys()` function.

Declaration

```
python {
       def <name_of_the_python_function>(<log_message>, <optional_other_
arguments>):
             # <your-python-code>
             return <value_of_the_template_function>
};

template <template-name> {
       template($(python <name_of_the_python_function>));
};
```

Example: Writing template functions in Python

The following example creates a Python template function called `return_message` that returns the MESSAGE part of the log message.

```
@version: 7.0

python {
      def return_message(log_message):
            return log_message['MESSAGE']
};

destination d_local {
      file("/tmp/logs.txt" template("[$(python return_message)]\n"));
};
```

The following example creates a Python template function called `resolve_host` that receives an IP address as an argument, and attempts to resolve it into a hostname.

```
@version: 7.0

python {
      import socket

      def resolve_host(log_message, hostname):
            try:
                  return socket.gethostbyaddr(hostname)[0]
            except (socket.herror, socket.error):
                  return 'unknown'
};

destination d_local {
      file("/tmp/logs.txt" template("${ISODATE} $(python resolve_host
${SOURCE_IP}) ${MESSAGE}\n"));
};
```

replace-delimiter

Syntax:

```
$(replace-delimiter "<old-delimiters>" "<new-delimiter>" "<macro>")
```

Description: Replaces the delimiter character with a new one. For example, the following example replaces the tabulators (`\t`) in the message with semicolons (`;`):

```
$(replace-delimiter "\t" ";" "${MESSAGE}")
```

Available in syslog-ng PE3.57.0 and later.

sanitize

Syntax:

```
$(sanitize <options> "<macro1>" "<macro2> ...")
```

Description: This file replaces the special characters in macro values, for example, it can replace the slash (/) characters in a filename with the underscore (_) character. If you specify multiple arguments, they will be concatenated using the / character, so they can be used as separate directory levels when used in filenames.

The function has the following options:

- `--ctrl-chars or -c`
- Filter control characters (characters that have an ASCII code of 32 or lower). This option is used by default.
- `--invalid-chars <characterlist> or -i <characterlist>`
- The list of characters to be replaced with underscores (_). The default list contains the / character. The following example replaces the \ and @ characters, so for example, fo\o@bar becomes foobar:

  ```
  $(sanitize -i \@ $PROGRAM)
  ```

- `--no-ctrl-chars or -C`
- Do not filter the control characters (characters that have an ASCII code of 32 or lower).
- `--replacement <replacement-character> or -r <replacement-character>`
- The character used to replace invalid characters. By default, this is the underscore (_). The following example replaces invalid characters with colons instead of underscores, so for example, `foo/bar` becomes `foo;bar`:

  ```
  $(sanitize -r ; $PROGRAM)
  ```

Example: Using the sanitize template function

The following example uses the sanitize function on two macros, and the results are used as directory names in a file destination.

```
file("/var/log/$(sanitize $HOST $PROGRAM)/messages");
```

This is equivalent to `file("/var/log/$HOST/$PROGRAM/messages");`, but any slashes in the values of the $HOST and $PROGRAM macros are replaced with underscores.

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}")
```

substr

Syntax:

```
$(substr "<argument>" "<offset>" "<length>")
```

Description: This function extracts a substring of a string.

- `argument`
- The string to extract the substring from, for example, `"${MESSAGE}"`
- `offset`
- Specifies where the substring begins (in characters). `0` means to start from the beginning of the string, `5` means to skip the first 5 characters of the string, and so on. Use negative numbers to specify where to start from the end of the string, for example, `-1` means the last character, `-5` means to start five characters before the end of the string.
- `length`
- (Optional parameter): The number of characters to extract. If not specified, the substring will be extracted from the offset to the end of the string. Use negative numbers to stop the substring before the end of the string, for example, `-5` means the substring ends five characters before the end of the string.

Example: Using the substr template function

Skip the first 15 characters of the message, and select the rest:

```
$(substr "${MESSAGE}" "15");
```

Select characters 16-30 of the message (15 characters with offset 15):

```
$(substr "${MESSAGE}" "15" "15");
```

Select the last 15 characters of the message:

```
$(substr "${MESSAGE}" "-15");
```

A template that converts the message to RFC3164 (BSD-syslog) format and truncates the messages to 1023 characters:

```
template t_truncate_messages {
      template("$(substr \"<$PRI>$DATE $HOST $MSGHDR$MESSAGE\" \"0\"
\"1023\")\n");
      template-escape(no);
 };
```

uppercase

Syntax:

```
$(uppercase "<macro>")
```

Description: Returns the uppercase version of the specified string or macro. For example, the following example uses the uppercase version of the hostname in a directory name:

```
destination d_file {
 file ("/var/log/${MONTH}/${DAY}/$(uppercase "${HOST}")/messages");
 };
```

Available in syslog-ng PE3.57.0 and later.

url-decode

Syntax:

```
$(url-decode <string-pr-macro-1> <string-pr-macro-2> ... )
```

Description: You can use the `url-decode` template function to decode url-encoded strings and macros. For example, `$(url-decode %3C%3E)` yields <>. The `url-decode` can receive multiple parameters (maximum 64). In this case, each parameter is decoded separately, and simply concatenated.

Available in syslog-ng PE version 3.187.0.11 and later.

url-encode

Syntax:

```
$(url-encode ${MESSAGE} )\n")
```

Description: You can use the `url-encode` template function together with the `telegram()` destination to send syslog messages to Telegram. The `url-encode` template function escapes strings. All input characters that are not a-z, A-Z, 0-9, '-', '.', '_' or '~' are converted to their "URL escaped" version.

Available in syslog-ng PE version 3.187.0.11 and later. (In version 3.16-3.17, this template function was called `urlencode`.)

uuid

Syntax:

```
$(uuid)
```

Description: Generates a Universally Unique IDentifier (UUID) that complies with RFC4122. That way, an UUID can be added to the message soon after it is received, so messages stored in multiple destinations can be identified. For example, when storing messages in a database and also in files, the UUID can be used to find a particular message both in the database and the files.

To generate a UUID, you can use a rewrite rule to create a new value-pair for the message.

Example: Using Universally Unique Identifiers

The following example adds a value-pair called `MESSAGE_UUID` to the message using a rewrite rule and a template.

```
rewrite r_add_uuid { set("$(uuid)" value("MESSAGE_UUID")); };

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

log { source(s_network);
         rewrite(r_add_uuid);
         destination(d_file);
};
```

NOTE: This template function is available only if the `tfuuid` module has been loaded.

By default, syslog-ng PE loads every available module. For details, see Loading modules.

Modifying the on-the-wire message format

Macros, templates, and template functions allow you to fully customize the format of the message. This flexibility makes it possible to use syslog-ng PE in some unexpected way if needed, for example, to emulate simple, plain-text protocols. The following example shows you how to send LPUSH commands to a Redis server.

NOTE: The purpose of this example is to demonstrate the flexibility of syslog-ng PE. A dedicated Redis destination is available in syslog-ng PE version 3.5. For details, see Storing name-value pairs in Redis.

The following template is a valid LPUSH command in accordance with the Redis protocol, and puts the $MESSAGE into a separate list for every $PROGRAM:

```
template t_redis_lpush {
      template("*3\r\n$$5\r\nLPUSH\r\n$$$(length
${PROGRAM})\r\n${PROGRAM}\r\n$$$(length ${MESSAGE})\r\n${MESSAGE}\r\n");
 };
```

If you use this template in a `network()` destination, syslog-ng PE formats the message according to the template, and sends it to the Redis server.

```
destination d_redis_tcp {
      network("127.0.0.1" port(6379) template(t_redis_lpush));
 };
```

Modifying messages using rewrite rules

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 parser: Parse and segment 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 Hard versus 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 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. Note that the `store-matches` flag is automatically enabled in rewrite rules.

Example: 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: 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("MESSAGE"));};
```

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 parser: Parse and segment structured messages and Processing message content with a pattern database). Hard macros cannot be modified. For details on the hard and soft macros, see Hard versus 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: 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 Creating custom SDATA fields.

You can also use the following options in rewrite rules that use the `set()` operator.

```
rewrite <name_of_the_rule> {
      set("<string to include>", value(<field name>), on-error("fallback-to-string");
};
```

Unsetting message fields

You can unset a macro or a field of the message, including any user-defined macros created using parsers (for details, see parser: Parse and segment structured messages and Processing message content with a pattern database). Hard macros cannot be modified. For details on hard and soft macros, see Hard versus soft macros). Note that the unset operation completely deletes any previous value of the field that you apply it on. Use the following syntax:

Declaration

```
rewrite <name_of_the_rule> {
      unset(value("<field name>"));
};
```

Example: Unsetting a message field

The following example unsets the HOST field of the message.

```
rewrite r_rewrite_unset{unset(value("HOST"));};
```

To unset a group of fields, you can use the `groupunset()` rewrite rule.

Declaration

```
rewrite  <name_of_the_rule>  {
       groupunset(values("<expression-for-field-names>"));
 };
```

Example: Unsetting a group of fields

The following rule clears all SDATA fields:

```
rewrite r_rewrite_unset_SDATA{ groupunset(values(".SDATA.*"));};
```

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 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: 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"));
 };
```

It is also possible to set the value of a field that does not exist yet, and create a new, custom name-value pair that is associated with the message. The following example creates the `.SDATA.groupID.fieldID@18372.4` field and sets its value to `yes`. If you use the `${.SDATA.groupID.fieldID@18372.4}` macro in a template or SQL table, its value will be `yes` for every message that was processed with this rewrite rule, and empty for every other message.

```
rewrite r_yes {
       set("yes" value(".SDATA.groupID.fieldID@18372.4"));
 };
```

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.

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: 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 `HOST` `FULLHOST` 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 F13.2, 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 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.

How conditional rewriting works

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);};
```

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: 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)));};
```

Anonymizing credit card numbers

Log messages of banking and e-commerce applications might include credit card numbers (Primary Account Number or PAN). According to privacy best practices and the requirements of the Payment Card Industry Data Security Standards (PCI-DSS), PAN must be rendered unreadable. The syslog-ng PE application uses a regular expression to detect credit card numbers, and provides two ways to accomplish this: you can either mask the credit card numbers, or replace them with a hash. To mask the credit card numbers, use the `credit-card-mask()` or the `credit-card-hash()` rewrite rules in a log path.

Usage

```
@include "scl/rewrite/cc-mask.conf"

rewrite { credit-card-mask(value("<message-field-to-process>")); };
```

By default, these rewrite rules process the MESSAGE part of the log message.

credit-card-hash()

Synopsis: credit-card-hash(value("<message-field-to-process>"))

Description: Process the specified message field (by default, `${MESSAGE}`), and replace any credit card numbers (Primary Account Number or PAN) with its 16-character-long SHA-1 hash.

credit-card-mask()

Synopsis: credit-card-mask(value("<message-field-to-process>"))

Description: Process the specified message field (by default, `${MESSAGE}`), and replace the 7-12th character of any credit card numbers (Primary Account Number or PAN) with asterisks (`*`). For example, syslog-ng PE replaces the number `5542043004559005` with `554204******9005`.

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 PCRE-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:

- Perl Compatible Regular Expressions (PCRE)
- Literal string searches
- Glob patterns without regular expression support

pcre

Description: Use Perl Compatible Regular Expressions (PCRE). Starting with syslog-ng PE version 3.1, PCRE expressions are supported on every platform. If the `type()` parameter is not specified, syslog-ng uses PCRE regular expressions by default.

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: Using PCRE regular expressions

```
rewrite r_rewrite_subst
        {subst("a*", "?", value("MESSAGE") 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
- 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: 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"); };
```

13

parser: Parse and segment structured messages

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

The syslog-ng PE application provides the following possibilities to parse the messages, or parts of the messages:

- By default, syslog-ng PE parses every message as a syslog message. To disable message parsing, use the `flags(no-parse)` option of the source. To explicitly parse a message as a syslog message, use the `syslog` parser. For details, see Parsing syslog messages.
- To segment a message into columns using a CSV-parser, see 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 Parsing key=value pairs.
- To parse JSON-formatted messages, see JSON parser.
- To parse XML-formatted messages, see XML parser.
- To identify and parse the messages using a pattern database, see Processing message content with a pattern database.
- To parse a specially-formatted date or timestamp, see Parsing dates and timestamps.
- To write a custom parser in Python or Hy, see Python parser.

The syslog-ng PE application provides built-in parsers for the following application logs:

- Apache HTTP server access logs. For details, see The Apache Access Log Parser.
- Cisco devices. For details, see Cisco Parser.
- Messages formatted using the enterprise-wide message model (EWMM) of syslog-ng PE. For details, see Parsing enterprise-wide message model (EWMM) messages.
- Iptables logs. For details, see iptables parser.

- Linux Audit (`auditd`) logs. For details, see Linux audit parser.
- sudo logs. For details, see Sudo parser.

Parsing syslog messages

By default, syslog-ng PE parses every message using the `syslog-parser` as a syslog message, and fills the macros with values of the message. The `syslog-parser` does not discard messages: the message cannot be parsed as a syslog message, the entire message (including its header) is stored in the $MSG macro. If you do not want to parse the message as a syslog message, use the `flags(no-parse)` option of the source.

You can also use the `syslog-parser` to explicitly parse a message, or a part of a message as a syslog message (for example, after rewriting the beginning of a message that does not comply with the syslog standards).

Example: Using junctions

For example, suppose that you have a single network source that receives log messages from different devices, and some devices send messages that are not RFC-compliant (some routers are notorious for that). To solve this problem in earlier versions of syslog-ng PE, you had to create two different network sources using different IP addresses or ports: one that received the RFC-compliant messages, and one that received the improperly formatted messages (for example, using the `flags(no-parse)` option). Using junctions this becomes much more simple: you can use a single network source to receive every message, then use a junction and two channels. The first channel processes the RFC-compliant messages, the second everything else. At the end, every message is stored in a single file. The filters used in the example can be `host()` filters (if you have a list of the IP addresses of the devices sending non-compliant messages), but that depends on your environment.

```
log {
      source { syslog(ip(10.1.2.3) transport("tcp") flags(no-parse));
};
      junction {
            channel { filter(f_compliant_hosts); parser { syslog-parser
(); }; };
            channel { filter(f_noncompliant_hosts); };
      };
      destination { file("/var/log/messages"); };
};
```

Since every channel receives every message that reaches the junction, use the `flags(final)` option in the channels to avoid the unnecessary processing the messages multiple times:

```
log {
      source { syslog(ip(10.1.2.3) transport("tcp") flags(no-parse));
};
      junction {
            channel { filter(f_compliant_hosts); parser { syslog-parser
(); }; flags(final); };
            channel { filter(f_noncompliant_hosts); flags(final); };
      };
      destination { file("/var/log/messages"); };
};
```

Note that syslog-ng PE has several parsers that you can use to parse non-compliant messages. You can even write a custom syslog-ng parser in Python. For details, see parser: Parse and segment structured messages.

Note that by default, the `syslog-parser` attempts to parse the message as an RFC3164-formatted (BSD-syslog) message. To parse the message as an RFC5424-formatted message, use the `flags(syslog-protocol)` option in the parser.

```
syslog-parser(flags(syslog-protocol));
```

Options of `syslog-parser` parsers

The `syslog-parser` 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)`

flags()

Type:	assume-utf8, empty-lines, expect-hostname, guess-timezone, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, store-raw-message, 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.
- *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.
- *guess-timezone*: Attempt to guess the timezone of the message if this information is not available in the message.
- *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 `file()` and `pipe()` 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 MESSAGE part of the syslog message (available using the `${MESSAGE}` macro). 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)`.

- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if 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.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng PE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${RAWMSG}` macro. You can forward this raw message in its original form to another syslog-ng node using the syslog-ng() destination, or to a SIEM system, ensuring that the SIEM can process it. Available only in 7.0.93.16 and later.
- *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 IETF-syslog messages). If theBOM[1]character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

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}`).

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

1
The byte order mark (BOM) is a Unicode character used to signal the byte-order of the message text.

to achieve this is to use a csv (comma-separated-values) parser (for other methods and possibilities, see the other sections of parser: Parse and segment 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_
strings>"))
            );
        };
```

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

Example: 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: 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.mycompany
```

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: 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.SEC",
"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: 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 parser: Parse and segment 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_
strings>"))
            );
        };
```

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

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.

String delimiters:

If you have to use a string as a delimiter, list your string delimiters in the `delimiters(strings("<delimiter_string1>", "<delimiter_string2>", ...)")` format.

By default, syslog-ng PE uses space as a delimiter. If you want to use only the strings as delimiters, you have to disable the space delimiter, for example: `delimiters(chars(""), strings("<delimiter_string>"))`

Otherwise, syslog-ng PE will use the string delimiters in addition to the default character delimiter, so `delimiters(strings("=="))` actually equals `delimiters(chars(" "), strings("=="))`, and not `delimiters(chars(""), strings("=="))`

Multiple delimiters:

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.

dialect()

Synopsis: escape-none|escape-backslash|escape-double-char

Description: Specifies how to handle escaping in the parsed message. The following values are available. Default value: `escape-none`

- *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.

```
parser p_demo_parser {
    csv-parser(
        prefix(".csv.")
        delimiters(" ")
```

```
        dialect(escape-backslash)
        flags(strip-whitespace, greedy)
        columns("column1", "column2", "column3")
    );
};
```

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 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: 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: Parsing Apache log files.

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

This parser does not have a default prefix. To configure a custom prefix, use the following format:

```
parser  {
       csv-parser(prefix("myprefix."));
 };
```

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}`).

For examples, see Example: Segmenting hostnames separated with a dash and Example: 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. The syslog-ng PE application automatically trims any leading or trailing whitespace characters from the keys and values, and also parses values that contain unquoted whitespace. For details on using value-pairs in syslog-ng PE see 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 are 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 Hard versus 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: 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.

extract-stray-words-into()

Synopsis: extract-stray-words-into("<name-value-pair>")

Description: Specifies the name-value pair where syslog-ng PE stores any stray words that appear before or between the parsed key-value pairs (mainly when the `pair-separator()` option is also set). If multiple stray words appear in a message, then syslog-ng PE stores them as a comma-separated list. Note that the `prefix()` option does not affect the name-value pair storing the stray words. Default value: `N/A`

Example: Extracting stray words in key-value pairs

For example, consider the following message:

```
VSYS=public; Slot=5/1; protocol=17; source-ip=10.116.214.221; source-
port=50989; destination-ip=172.16.236.16; destination-
port=162;time=2016/02/18 16:00:07; interzone-emtn_s1_vpn-enodeb_om;
inbound; policy=370;
```

This is a list of key-value pairs, where the value separator is `=` and the pair separator is `;`. However, before the last key-value pair (`policy=370`), there are two stray words: `interzone-emtn_s1_vpn-enodeb_om inbound`. If you want to store or process these, specify a name-value pair to store them in the `extract-stray-words-into()` option, for example, `extract-stray-words-into("my-stray-words")`. The value of `${my-stray-words}` for this message will be `interzone-emtn_s1_vpn-enodeb_om, inbound`

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `kv-parser()` uses the `.kv.` prefix. To modify it, use the following format:

```
parser  {
       kv-parser(prefix("myprefix."));
 };
```

pair-separator()

Synopsis: pair-separator("<separator-string>")

Description: Specifies the character or string that separates the key-value pairs from each other. Default value: , (a comma followed by a whitespace)

For example, to parse `key1=value1;key2=value2` pairs, use `kv-parser(pair-separator(";"));`

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("<separator-character>")

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(":"));`

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 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 Hard versus 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: 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 {
        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]"));` parser is equivalent to the `msg.foo.bar[5]` javascript code. Note that the resulting expression must be a JSON object in order to 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]`.

In addition to alphanumeric characters, the key of the JSON object can contain the following characters: `!"#$%&'()*+,-/:;<=>?@\^_`{|}~`

It cannot contain the following characters: `.[]`

Example: Convert logstash eventlog format v0 to v1

The following parser converts messages in the logstash eventlog v0 format to the v1 format.

```
parser p_jsoneventv0 {
  channel {
    parser { json-parser(extract-prefix("@fields")); };
    parser { json-parser(prefix(".json.")); };
    rewrite {
```

```
      set("1" value("@version"));
      set("${.json.@timestamp}" value("@timestamp"));
      set("${.json.@message}" value("message"));
    };
  };
};
```

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: 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 value-pairs().)

```
parser  {
            json-parser(
                  marker("@cee:")
                  prefix(".cee.")
            );
      };
```

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

This parser does not have a default prefix. To configure a custom prefix, use the following format:

```
parser  {
       json-parser(prefix("myprefix."));
 };
```

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}`).

XML parser

Extensible Markup Language (XML) is a text-based open standard designed for both human-readable and machine-readable data interchange. Like JSON, it is used primarily to transmit data between a server and web application. It is described in W3C Recommendation: Extensible Markup Language (XML).

The XML parser processes input in XML format, and adds the parsed data to the message object.

To create an XML parser, define an `xml_parser` that has the `xml()` option. By default, the parser will process the `${MESSAGE}` part of the log message. To process other parts of a log message using the XML parser, use the `template()` option. You can also define the parser inline in the log path.

Declaration

```
parser  xml_name  {
       xml(template()
             prefix()
             drop-invalid()
             exclude-tags()
             strip-whitespaces()
       );
 };
```

Example: Using an XML parser

In the following example, the source is an XML-encoded log message. The destination is a file that uses the `format-json` template. The log line connects the source, the destination and the parser.

```
source  s_local  {
             file("/tmp/aaa");
 };

 destination  d_local  {
       file("/tmp/bbb"  template("$(format-json  .xml.*)\n"));
 };

 parser  xml_parser  {
           xml();
 };

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

You can also define the parser inline in the log path.

```
log  {
       source(s_file);
       parser  {  xml(prefix(".SDATA"));  };
       destination(d_file);
 };
```

The XML parser inserts an "`.xml`" prefix by default before the extracted name-value pairs. Since `format-json` replaces a dot with an underscore at the beginning of keys, the "`.xml`" prefix becomes "`_xml`". Attributes get an _ prefix. For example, from the XML input:

```
<tags attr='attrval'>part1<tag1>Tag1 Leaf</tag1>part2<tag2>Tag2
Leaf</tag2>part3</tags>
```

The following output is generated:

```
{"_xml":{"tags":{"tag2":"Tag2 Leaf","tag1":"Tag1 Leaf","_
attr":"attrval","tags":"part1part2part3"}}}
```

When the text is separated by tags on different levels or tags on the same level, the parser uses the list-handling functionality (enabled by default) to handle lists in the XML.

The list-handling functionality of the XML parser separates vector-like structures by a comma as separate entries. Using the following structure as an example:

```
<vector>
       <entry>value1</entry>
       <entry>value  2</entry>
       <entry>Doe,John</entry>
       <entry>value3</entry>
       ...
       <entry>valueN</entry>
 </vector>
```

After parsing, the entries are separated by a comma. If an entry has a space or is separated by a comma, for example, **value 2** or **Doe,John** in the previous example, quoting is applied to the entry:

```
vector.entry = value1,"value 2","Doe,John",value3...valueN
```

Note that if you disable the list-handling functionality, the XML parser cannot address each element of a vector-like structure individually. Using the following structure as an example:

```
<vector>
       <entry>value1</entry>
       <entry>value2</entry>
       ...
       <entry>valueN</entry>
 </vector>
```

After parsing, the entries are not addressed individually. Instead, the text of the entries are concatenated:

```
vector.entry = "value1value2...valueN"
```

For more information about the list-handling functionality, see Limitations of the XML parsers.

Whitespaces are kept as they are in the XML input. No collapsing happens on significant whitespaces. For example, from this input XML:

```
<133>Feb 25 14:09:07 webserver syslogd: <b>|Test\n\n   Test2|</b>\n
```

The following output is generated:

```
[2017-09-04T13:20:27.417266] Setting value; msg='0x7f2fd8002df0', name='.xml.b',
value='|Test\x0a\x0a   Test2|'
```

However, note that users can choose to strip whitespaces using the `strip-whitespaces()` option.

Configuration hints

Define a source that correctly detects the end of the message, otherwise the XML parser will consider the input invalid, resulting in a parser error.

To ensure that the end of the XML document is accurately detected, use any of the following options:

- Ensure that the XML is a single-line message.
- In the case of multiline XML documents:
  - If the opening and closing tags are fixed and known, you can use `multi-line-mode(prefix-suffix)`. Using regular expressions, specify a prefix and suffix matching the opening and closing tags. For details on using `multi-line-mode(prefix-suffix)`, see the `multi-line-prefix()` and `multi-line-suffix()` options.
  - In the case of TCP, you can encapsulate and send the document in syslog-protocol format, and use a `syslog()` source. Make sure that the message conforms to the octet counting method described in RFC6587.

    For example:

    ```
    59 <133>Feb 25 14:09:07 webserver syslogd: <book>\nText\n</book>
    ```

    Considering the new lines as one character, `59` is appended to the original message.
  - You can use a datagram-based source. In the case of datagram-based sources, the protocol signals the end of the message automatically. Ensure that the complete XML document is written in one message.
  - Unless the opening and closing tags are fixed and known, stream-based sources are currently not supported.

In case you experience issues, start syslog-ng with debug logs enabled. There will be a debug log about the incoming log entry, which shows the complete message to be parsed. The entry should contain the entire XML document.

NOTE: If your log messages are entirely in .xml format, make sure to disable any message parsing on the source side by including the `flags("no-parse")` option in your source statement. This will put the entire log message in the `$MESSAGE` macro, which is the field that the XML parser parses by default.

Limitations of the XML parsers

The XML parser comes with certain limitations.

Using the list-handling functionality with vector-like structures

The XML parser uses the list-handling functionality to handle lists in the XML. The list-handling functionality has limitations when handling name-value pairs or quoting in SDATA as described below. Note that you can disable the list-handling functionality if needed.

The list-handling functionality of the XML parser separates vector-like structures by a comma as separate entries. Using the following structure as an example:

```
<vector>
       <entry>value1</entry>
       <entry>value  2</entry>
       <entry>Doe,John</entry>
       <entry>value3</entry>
       ...
       <entry>valueN</entry>
 </vector>
```

After parsing, the entries are separated by a comma. If an entry has a space or is separated by a comma, for example, **value 2** or **Doe,John** in the previous example, quoting is applied to the entry:

```
vector.entry = value1,"value 2","Doe,John",value3...valueN
```

Using the list-handling functionality with name-value pairs

As every value in name-value pairs can be quoted, One Identity recommends that you access name-values as lists as follows:

- Use list-related template functions on the list created by the XML parser.
- Use type-hinting using the format-json template function as shown in the example below:

  ```
  template("$(format-json --scope dot-nv-pairs LIST=list
  (${.xml.Event.EventData.Data}))\n")
  ```

Using the list-handling functionality with SDATA

According to RFC5424, SDATA parameter values must be quoted with double-quote ('"') characters. If the value contains double-quotes, they must be escaped with a backslash (\) character.

Due to the list-handling functionality of the XML parser, parsed XML text contents are also quoted using double-quote ('"') characters. As parsed XML text content are part of the message, they are quoted when used as SDATA parameter values.

Using the following structure as an example:

```
<Event>
 <Data>42</Data>
 <Data>Testing  testing</Data>
 </Event>
```

The expected name-value pair is as follows:

```
Event.Data = 42,"Testing testing"
```

In SDATA, this is quoted as shown below:

```
[Event Data="42,\"Testing testing\""]
```

Disabling the list-handling functionality

To disable the list-handling functionality, use the `create_lists(yes/no)` option as shown below. The default value is set to `yes`.

```
parser  p_xml  {
       xml(create_lists(no));
 };
```

Note that if you disable the list-handling functionality, the XML parser cannot address each element of a vector-like structure individually. Using the following structure as an example:

```
<vector>
       <entry>value1</entry>
       <entry>value2</entry>
       ...
       <entry>valueN</entry>
 </vector>
```

After parsing, the entries are not addressed individually. Instead, the text of the entries are concatenated:

```
vector.entry = "value1value2...valueN"
```

CDATA

The XML parser does not support CDATA. CDATA inside the XML input is ignored. This is true for the processing instructions as well.

Inherited limitations

The XML parser is based on the glib XML subset parser, called "GMarkup" parser, which is not a full-scale XML parser. It is intended to parse a simple markup format that is a subset of XML. Some limitations are inherited:

- Do not use the XML parser if you expect to interoperate with applications generating full-scale XML. Instead, use it for application data files, configuration files, log files, and so on, where you know your application will be the only one writing the file.
- The XML parser is not guaranteed to display an error message in the case of invalid XML. It may accept invalid XML. However, it does not accept XML input that is not well-formed (a condition that is weaker than requiring XML to be valid).

No support for long keys

If the key is longer than 255 characters, syslog-ng drops the entry and an error log is emitted. There is no chunking or any other way of recovering data, not even partial data. The entry will be replaced by an empty string.

Options of the XML parsers

The XML parser has the following options.

create-lists()

Synopsis:	create-lists()
Format:	yes|no
Default:	yes
Mandatory:	no

Description: If set, the list-handling functionality of the XML parser separates vector-like structures by a comma as separate entries. For more information, see Limitations of the XML parsers.

drop-invalid

Synopsis:	drop-invalid()
Format:	yes|no
Default:	no
Mandatory:	no

Description: If set, messages with an invalid XML will be dropped entirely.

exclude-tags

Synopsis:	exclude-tags()
Format:	list of globs
Default:	None If not set, no filtering is done.
Mandatory:	no

Description: The XML parser matches tags against the listed globs. If there is a match, the given subtree of the XML will be omitted.

Example: Using `exclude_tags`

```
parser  xml_parser  {
           xml(template("$MSG")  exclude_tags("tag1",  "tag2",  "inner*"));
 };
```

From this XML input:

```
<tag1>Text1</tag1><tag2>Text2</tag2><tag3>Text3<innertag>TextInner</innert
ag></tag3>
```

The following output is generated:

```
{"_xml":{"tag3":"Text3"}}
```

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 Hard versus

soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

The `prefix()` option is optional and its default value is `".xml"`.

strip-whitespaces

Synopsis:	strip-whitespaces()
Format:	yes|no
Default:	no
Mandatory:	no

Description: Strip the whitespaces from the XML text nodes before adding them to the message.

Example: Using `strip-whitespaces`

```
parser  xml_parser  {
           xml(template("$MSG")  strip_whitespaces(yes));
 };
```

From this XML input:

```
<tag1> Tag </tag1>
```

The following output is generated:

```
{"_xml":{"tag1":"Tag"}}
```

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}`).

Parsing dates and timestamps

The date parser can extract dates from non-syslog messages. It operates by default on the ${MSG} part of the log message, but can operate on any template or field provided. The

parsed date will be available as the sender date (that is, the ${S_DATE}, ${S_ISODATE}, ${S_MONTH}, and so on, and related macros). (To store the parsed date as the received date, use the `timestamp(recvd)` option.)

NOTE: Note that parsing will fail if the format string does not match the entire template or field. Since by default syslog-ng Premium Edition (syslog-ng PE) uses the ${MESSAGE} part of the log message, parsing will fail, unless the log message contains only a date, but that is unlikely, so practically you will have to segment the message (for example, using a csv-parser()) before using the `date-parser()`. You can also use `date-parser()` to parse dates received in a JSON or key-value-formatted log message.

Declaration

```
parser parser_name {
       date-parser(
             format("<format-string-for-the-date>")
             template("<field-to-parse>'")
       );
 };
```

Example: Using the date-parser()

In the following example, syslog-ng PE parses dates like `01/Jan/2016:13:05:05 PST` from a field called `MY_DATE` using the following format string: `format ("%d/%b/%Y:%H:%M:%S %Z")` (how you create this field from the incoming message is not shown in the example). In the destination template every message will begin with the timestamp in ISODATE format. Since the syslog parser is disabled, syslog-ng PE will include the entire original message (including the original timestamp) in the ${MESSAGE} macro.

```
source s_file {
      file("/tmp/input" flags(no-parse));
};

destination d_file {
      file( "/tmp/output" template("${S_ISODATE} ${MESSAGE}\n") );
};

log {
      source(s_file);
      parser { date-parser(format("%d/%b/%Y:%H:%M:%S %Z") template
("${MY_DATE}")); };
      destination(d_file);
};
```

In the template option, you can use template functions to specify which part of the message to parse with the format string. The following example selects the first 24 characters of the ${MESSAGE} macro.

```
date-parser(format("%d/%b/%Y:%H:%M:%S  %Z")  template("$(substr  ${MSG}  0
24)")  );
```

In syslog-ng PE version 7.0.18 and later, you can specify a comma-separated list of formats to parse multiple date formats with a single parser. For example:

```
date-parser(format(
       "%FT%T.%f",
       "%F  %T,%f",
       "%F  %T"
 ));
```

Options of `date-parser()` parsers

The `date-parser()` parser has the following options.

flags()

Type:	guess-timezone
Default:	empty string

guess-timezone: Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp. For example:

```
date-parser(flags(guess-timezone));
```

format()

Synopsis:	format(string)
Default:	

Description: Specifies the format how syslog-ng PE should parse the date. You can use the following format elements:

```
%%      PERCENT
%a      day of the week, abbreviated
%A      day of the week
%b      month abbr
%B      month
%c      MM/DD/YY HH:MM:SS
%C      ctime format: Sat Nov 19 21:05:57 1994
%d      numeric day of the month, with leading zeros (eg 01..31)
%e      like %d, but a leading zero is replaced by a space (eg  1..31)
%f      microseconds, leading 0's, extra digits are silently discarded
%D      MM/DD/YY
%G      GPS week number (weeks since January 6, 1980)
%h      month, abbreviated
%H      hour, 24 hour clock, leading 0's)
%I      hour, 12 hour clock, leading 0's)
%j      day of the year
%k      hour
%l      hour, 12 hour clock
%L      month number, starting with 1
%m      month number, starting with 01
%M      minute, leading 0's
%n      NEWLINE
%o      ornate day of month -- "1st", "2nd", "25th", etc.
%p      AM or PM
%P      am or pm (Yes %p and %P are backwards :)
%q      Quarter number, starting with 1
%r      time format: 09:05:57 PM
%R      time format: 21:05
%s      seconds since the Epoch, UCT
%S      seconds, leading 0's
%t      TAB
%T      time format: 21:05:57
%U      week number, Sunday as first day of week
%w      day of the week, numerically, Sunday == 0
%W      week number, Monday as first day of week
%x      date format: 11/19/94
%X      time format: 21:05:57
%y      year (2 digits)
%Y      year (4 digits)
%Z      timezone in ascii. eg: PST
%z      timezone in format -/+0000
```

For example, for the date `01/Jan/2016:13:05:05 PST` use the following format string: `format("%d/%b/%Y:%H:%M:%S %Z")`

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}`).

time-stamp()

Synopsis:	stamp | recvd
Default:	stamp

Description: Determines if the parsed date values are treated as sent or received date. If you use `time-stamp()`, syslog-ng PE adds the parsed date to the S_ macros (corresponding to the sent date). If you use `time-zone(recvd)`, syslog-ng PE adds the parsed date to the R_ macros (corresponding to the received date).

time-zone()

Synopsis:	timezone(string)
Default:	

Description: If this option is set, syslog-ng PE assumes that the parsed timestamp refers to the specified timezone. The timezone set in the `time-zone()` option overrides any timezone information parsed from the timestamp.

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.

Cisco Parser

The Cisco Parser can parse the log messages of various Cisco devices. The messages of these devices often do not completely comply with the syslog RFCs, making them difficult to parse. The `cisco-parser()` of syslog-ng PE solves this problem, and can separate these log messages to name-value pairs, extracting also the Cisco-specific values, for example, the mnemonic. For details on using value-pairs in syslog-ng PE see Structuring macros, metadata, and other value-pairs. The parser can parse variations of the following message format:

```
<pri>(sequence: )?(origin-id: )?(timestamp? timezone?: )?%msg
```

For example:

```
<189>29: foo: *Apr 29 13:58:40.411: %SYS-5-CONFIG_I: Configured from console by
console
<190>30: foo: *Apr 29 13:58:46.411: %SYS-6-LOGGINGHOST_STARTSTOP: Logging to
host 192.168.1.239 stopped - CLI initiated
<190>31: foo: *Apr 29 13:58:46.411: %SYS-6-LOGGINGHOST_STARTSTOP: Logging to
host 192.168.1.239 started - CLI initiated
<189>32: 0.0.0.0: *Apr 29 13:59:12.491: %SYS-5-CONFIG_I: Configured from
console by console
```

Note that not every Cisco log message conforms to this format. If you find a message that the `cisco-parser()` cannot properly parse, send it to documentation@balabit.com so we can improve the parser.

The syslog-ng PE application normalizes the parsed log messages into the following format:

```
${MESSAGE}=%FAC-SEV-MNEMONIC: message
${HOST}=origin-id
```

By default, the Cisco-specific fields are extracted into the following name-value pairs:`${.cisco.facility}`, `${.cisco.severity}`, `${.cisco.mnemonic}`. You can change the prefix using the `prefix` option.

Declaration

```
@version: 7.0
@include "scl.conf"
log {
      source { udp(flags(no-parse)); };
      parser { cisco-parser(); };
      destination { ... };
};
```

Note that you have to disable message parsing in the source using the `flags(no-parse)` option for the parser to work.

The `cisco-parser()` is actually a reusable configuration snippet configured to parse Cisco messages. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of this configuration snippet on GitHub.

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `cisco-parser()` uses the `.cisco.` prefix. To modify it, use the following format:

```
parser  {
       cisco-parser(prefix("myprefix."));
 };
```

Linux audit parser

The Linux audit parser can parse the log messages of the Linux audit subsystem (`auditd`). The syslog-ng PE application can separate these log messages to name-value pairs. For details on using value-pairs in syslog-ng PE see Structuring macros, metadata, and other value-pairs. The `linux-audit-parser()` is available in version 7.0.133.7 and later.

The following is a sample log message of `auditd`:

```
type=SYSCALL msg=audit(1441988805.991:239): arch=c000003e syscall=59 success=yes
exit=0 a0=7fe49a6d0e98 a1=7fe49a6d0e40 a2=7fe49a6d0e80 a3=2 items=2 ppid=3652
pid=3660 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=
(none) ses=5 comm="dumpe2fs" exe="/sbin/dumpe2fs" key=(null)
type=EXECVE msg=audit(1441988805.991:239): argc=3 a0="dumpe2fs" a1="-h"
a2="/dev/sda1"
type=CWD msg=audit(1441988805.991:239):  cwd="/"
type=PATH msg=audit(1441988805.991:239): item=0 name="/sbin/dumpe2fs"
inode=137078 dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL
type=PATH msg=audit(1441988805.991:239): item=1 name="/lib64/ld-linux-x86-
64.so.2" inode=5243184 dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
nametype=NORMAL
type=PROCTITLE msg=audit(1441988805.991:239):
proctitle=64756D7065326673002D68002F6465762F73646131
```

Certain fields of the audit log can be encoded in hexadecimal format, for example, the `arch` field, or the `a<number>` fields in the previous example. The syslog-ng PE application automatically decodes these fields (for example, the `c000003e` value becomes `x86_64`).

The syslog-ng PE application extracts every field into name-value pairs. It automatically decodes the following fields:

- name
- proctitle
- path
- dir
- comm
- ocomm
- data
- old
- new

To parse the log messages of the Linux audit subsystem, define a parser that has the `linux-audit-parser()` option. By default, the parser will process the `${MESSAGE}` part of the log message. To process other parts of a log message, use the `template()` option. You can also define the parser inline in the log path.

Declaration

```
parser  parser_name  {
       linux-audit-parser(
             prefix()
             template()
       );
 };
```

Example: Using the `linux-audit-parser()` parser

In the following example, the source is a log file created by auditd. Since the audit log format is not a syslog format, the syslog parser is disabled, so that syslog-ng PE does not parse the message: `flags(no-parse)`. The parser inserts `".auditd."` 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_auditd {
    file(/var/log/audit/audit.log flags(no-parse));
};

destination d_json {
    file(
        "/tmp/test.json"
```

```
        template("$(format-json .auditd.*)\n")
    );
};

parser p_auditd {
    linux-audit-parser (prefix(".auditd."));
};

log {
    source(s_auditd);
    parser(p_auditd);
    destination(d_json);
};
```

You can also define the parser inline in the log path.

```
source  s_auditd  {
       file(/var/log/audit/audit.log);
 };

 destination  d_json  {
       file(
             "/tmp/test.json"
             template("$(format-json  .auditd.*)\n")
       );
 };

 log  {
       source(s_auditd);
       parser  {
             linux-audit-parser  (prefix(".auditd."));
       };
       destination(d_json);
 };
```

Options of `linux-audit-parser()` parsers

The `linux-audit-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.

By default, `linux-audit-parser()` uses the `.auditd.` prefix. To modify it, use the following format:

```
parser  {
       linux-audit-parser(prefix("myprefix."));
 };
```

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}`).

Python parser

The Python log parser (available in syslog-ng PE version 3.107.0.2 and later) allows you to write your own parser in Python. Practically, that way you can process the log message (or parts of the log message) any way you need.

The following points apply to using Python blocks in syslog-ng PE in general:

- Only the default Python modules are available (that is, you cannot import external Python modules, and One Identity does not support using external Python modules).
- The syslog-ng PE application uses its own Python interpreter (shipped with the default syslog-ng PE installation) instead of the system's Python interpreter.
- The syslog-ng PE application is shipped with Python version 3.8.
- The Python block must be a top-level block in the syslog-ng PE configuration file.
- If you store the Python code in a separate Python file and only include it in the syslog-ng PE configuration file, make sure that the PYTHON_PATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start syslog-ng PE manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following

command: `export PYTHONPATH=/opt/syslog-ng/etc`

In production, when syslog-ng PE starts on boot, you must configure your startup script to include the Python path. The exact method depends on your operating system. For recent Red Hat Enterprise Linux, Fedora, and CentOS distributions that use systemd, the `systemctl` command sources the `/etc/sysconfig/syslog-ng` file before starting syslog-ng PE. (On openSUSE and SLES, `/etc/sysconfig/syslog` file.) Append the following line to the end of this file: `PYTHONPATH="<path-to-your-python-file>"`, for example, `PYTHONPATH="/opt/syslog-ng/etc"`

- The Python object is initiated every time when syslog-ng PE is started or reloaded.

  CAUTION:

  If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.

- The Python block can contain multiple Python functions.
- Using Python code in syslog-ng PE can significantly decrease the performance of syslog-ng PE, especially if the Python code is slow. In general, the features of syslog-ng PE are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The syslog-ng PE application does not do any of this.
- Python error messages are available in the `internal()` source of syslog-ng PE.
- You can access the name-value pairs of syslog-ng PE directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of syslog-ng PE. For details, see Logging from your Python code.

- **Support disclaimer**

  CAUTION:

  This is a Preview Feature, which provides an insight to planned enhancements to functionality in the product. Consider this Preview Feature a work in progress, as it may not represent the final design and functionality.

  This feature has completed QA release testing, but its full impact on production systems has not been determined yet, and potential future changes in functionality and the user interface may result in compatibility issues in your current settings.

  One Identity recommends the following:

  - **Consider the potential risks when using this functionality in a production environment.**
  - **Consider the Support Policy on Product Preview Features before using this functionality in a production environment.**
  - **Closely and regularly keep track of official One Identity announcements about potential changes in functionality and the user interface. If these potential changes affect your configuration, check the changes you have to make in your configuration, otherwise your syslog-ng PE application may not start after upgrade.**
  - **Always perform tests prior to upgrades in order to avoid the risks mentioned.**

  However, you are welcome to try this feature and if you have any feedback, Contact One Identity.

  Support Policy on Product Preview Features

  The One Identity Support Team will:

  - **Accept and review each service request opened regarding a Preview Feature.**
  - **Consider all service requests relating to a Preview Features as severity level 3.**
  - **Provide best effort support to resolve any issues relating to a Preview Feature.**
  - **Work with customers to log any product defects or enhancements relating to Preview Features.**
  - **Not accept requests for escalations regarding Preview Features.**
  - **Not provide after-hours support for Preview Features.**

  Using Python in syslog-ng PE is recommended only if you are familiar with both Python and syslog-ng PE. One Identity is not responsible for the quality, resource requirements, or any bugs in the Python code, nor any syslog-ng PE crashes,

message losses, or any other damage caused by the improper use of this feature, unless explicitly stated in a contract with One Identity.

Declaration

Python parsers consist of two parts. The first is a syslog-ng PE parser object that you use in your syslog-ng PE configuration, for example, in the log path. This parser references a Python class, which is the second part of the Python parsers. The Python class processes the log messages it receives, and can do virtually anything that you can code in Python.

```
parser <name_of_the_python_parser>{
    python(
       class("<name_of_the_python_class_executed_by_the_parser>")
    );
};

python {
class MyParser(object):
       def init(self, options):
              '''Optional. This method is executed when syslog-ng is started
or reloaded.'''
              return True
       def deinit(self):
              '''Optional. This method is executed when syslog-ng is stopped
or reloaded.'''
              pass
       def parse(self, msg):
              '''Required. This method receives and processes the log
message.'''
              return True
};
```

Methods of the python() parser

The `init (self, options)` method (optional)

The syslog-ng PE application initializes Python objects only when it is started or reloaded. That means it keeps the state of internal variables while syslog-ng PE is running. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your parser to work. For example, if you want to perform a lookup from a file or a database, you can open the file or connect to the database here, or you can initialize a counter that you will increase in the `parse()` method.

The return value of the `init()` method must be `True`. If it returns `False`, or raises an exception, syslog-ng PE will not start.

`options`: This optional argument contains the contents of the `options()` parameter of the parser object as a Python dict.

```
parser my_python_parser{
  python(
    class("MyParser")
    options("regex", "seq: (?P<seq>\\d+), thread: (?P<thread>\\d+), runid:
(?P<runid>\\d+), stamp: (?P<stamp>[^ ]+) (?P<padding>.*$)")
  );
};
class MyParser(object):
    def init(self, options):
        pattern = options["regex"]
        self.regex = re.compile(pattern)
        self.counter = 0
        return True
```

The `parse(self, log_message)` method

The `parse()` method processes the log messages it receives, and can do virtually anything that you can code in Python. This method is required, otherwise syslog-ng PE will not start.

The return value of the `parse()` method must be `True`. If it returns `False`, or raises an exception, syslog-ng PE will drop the message.

- To reference a name-value pair or a macro in the Python code, use the following format. For example, if the first argument in the definition of the function is called `log-message`, the value of the HOST macro is `log-message['HOST']`, and so on. (The `log-message` contains the entire log message (not just the text body) in a structure similar to a Python dict, but it is actually an object.)
- You can define new name-value pairs in the Python function. For example, if the first argument in the definition of the function is called `log-message`, you can create a new name-value pair like this: `log_message["new-macro-name"]="value"`. This is useful when you parse a part of the message from Python, or lookup a value based on data extracted from the log message.

  Note that the names of the name-value pairs are case-sensitive. If you create a new name-value pair called `new-macro-name` in Python, and want to reference it in another part of the syslog-ng PE configuration file (for example, in a template), use the `${new-macro-name}` macro.
- You cannot override hard macros (see Hard versus soft macros).
- To list all available keys (names of name-value pairs), use the `log_message.keys()` function.

The `deinit(self)` method (optional)

This method is executed when syslog-ng PE is stopped or reloaded.

CAUTION:

If you reload syslog-ng PE, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of syslog-ng PE typically involves a reload.

Example: Parse loggen logs

The following sample code parses the messages of the `loggen` tool (for details, see The loggen manual page). The following is a sample loggen message:

```
<38>2017-04-05T12:16:46 localhost prg00000[1234]: seq: 0000000000, thread:
0000, runid: 1491387406, stamp: 2017-04-05T12:16:46
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPA
DDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
```

The syslog-ng PE parser object references the LoggenParser class and passes a set of regular expressions to parse the loggen messages. The `init()` method of the LoggenParser class compiles these expressions into a pattern. The `parse` method uses these patterns to extract the fields of the message into name-value pairs. The destination template of the syslog-ng PE log statement uses the extracted fields to format the output message.

```
@version: 7.0
@include "scl.conf"
parser my_python_parser{
  python(
    class("LoggenParser")
    options("regex", "seq: (?P<seq>\\d+), thread: (?P<thread>\\d+), runid:
(?P<runid>\\d+), stamp: (?P<stamp>[^ ]+) (?P<padding>.*$)")
  );
};
log {
  source { tcp(port(5555)); };
  parser(my_python_parser);
  destination {  file("/tmp/regexparser.log.txt" template("seq: $seq
thread: $thread runid: $runid stamp: $stamp my_counter: $MY_COUNTER"));};
};
python {
import re
class LoggenParser(object):
    def init(self, options):
        pattern = options["regex"]
        self.regex = re.compile(pattern)
```

```
        self.counter = 0
        return True
    def deinit(self):
        pass
    def parse(self, log_message):
        match = self.regex.match(log_message['MESSAGE'])
        if match:
            for key, value in match.groupdict().items():
                log_message[key] = value
            log_message['MY_COUNTER'] = self.counter
            self.counter += 1
            return True
        return False
};
```

Example: Parse Windows eventlogs in Python - performance

The following example uses regular expressions to process Windows log messages received in XML format from the syslog-ng Agent for Windows application. The parser extracts different fields from messages received from the Security and the Application eventlog containers. Using the following configuration file, syslog-ng PE could process about 25000 real-life Windows log messages per second.

```
@version: 7.0
options {
    keep_hostname(yes);
    keep_timestamp(no);
    stats_level(2);
    use_dns(no);
};
source s_network_aa5fdf25c39d4017a8e504cdb641b477 {
    network(flags(no-parse)
        ip(0.0.0.0)
        log_fetch_limit(1000)
        log_iw_size(100000)
        max_connections(100)
        port(514));
};
parser p_python_parser_79c31da44bb64de6b5de84be4ae15a15 {
    python(options("regex_for_security", ".* Security ID:  (?P<security_
id>\\S+)   Account Name:  (?P<account_name>\\S+)   Account Domain:
```

```
(?P<account_domain>\\S+)   Logon ID:  (?P<logon_id>\\S+).*Process Name:
(?P<process_name>\\S+).*EventID (?P<event_id>\\d+)", "regex_others", "
(.*)EventID (?P<event_id>\\d+)")
class("EventlogParser"));
};
destination d_file_78363e1dd90c4ebcbb0ee1eff5a2e310 {
    file("/var/testdb_working_dir/fcd713a2-d48e-4025-9192-
ec4a9852cafa.$HOST"
        flush_lines(1000)
        log_fifo_size(200000));
};
log {
    source(s_network_aa5fdf25c39d4017a8e504cdb641b477);
    parser(p_python_parser_79c31da44bb64de6b5de84be4ae15a15);
    destination(d_file_78363e1dd90c4ebcbb0ee1eff5a2e310);

    flags(flow-control);
};

python {
import re
class EventlogParser(object):
    def init(self, options):
        self.regex_security = re.compile(options["regex_for_security"])
        self.regex_others = re.compile(options["regex_others"])
        return True
    def deinit(self):
        pass
    def parse(self, log_message):
        security_match = self.regex_security.match(log_message['MESSAGE'])
        if security_match:
            for key, value in security_match.groupdict().items():
                log_message[key] = value
        else:
            others_match = self.regex_others.match(log_message['MESSAGE'])
            if others_match:
                for key, value in others_match.groupdict().items():
                    log_message[key] = value
        return True
};
```

Parsing enterprise-wide message model (EWMM) messages

The `ewmm-parser()` can be used to parse messages sent by another syslog-ng host using the enterprise-wide message model (EWMM) format. Available in version 7.0.93.16 and later. Note that usually you do not have to use this parser directly, because the default-network-drivers() source automatically parses such messages.

Declaration

```
parser  parser_name  {
       ewmm-parser();
 };
```

Sudo parser

The sudo parser can parse the log messages of the sudo command. Available in version 7.0.93.16 and later.

Declaration

```
@version:  7.0
 @include  "scl.conf"
 log  {
       source  {  system();  };
       parser  {  sudo-parser();  };
       destination  {  ...  };
 };
```

The `sudo-parser()` is actually a reusable configuration snippet configured to parse sudo messages. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of this configuration snippet on GitHub.

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `sudo-parser()` uses the `.sudo.` prefix. To modify it, use the following format:

```
parser  {
       sudo-parser(prefix("myprefix."));
 };
```

iptables parser

The iptables parser can parse the log messages of the iptables command. Available in version 7.0.93.16 and later.

Declaration

```
@version:  7.0
 @include  "scl.conf"
 log  {
       source  {  system();  };
       parser  {  iptables-parser();  };
       destination  {  ...  };
 };
```

The `iptables-parser()` is actually a reusable configuration snippet configured to parse iptables messages. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of this configuration snippet on GitHub.

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 Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `iptables-parser()` uses the `.iptables.` prefix. To modify it, use the following format:

```
parser  {
       iptables-parser(prefix("myprefix."));
 };
```

14

Processing message content with a pattern database

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 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))?`

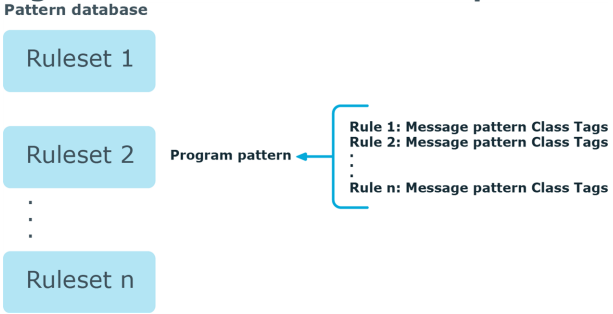
A syslog-ng database pattern for this message: `Accepted @QSTRING:auth_method: @for@QSTRING:username: @from @QSTRING:client_addr: @port @NUMBER:port:@ ssh2`

For details on using pattern databases to classify log messages, see Using pattern databases.

The structure of the pattern database

The pattern database is organized as follows:

Figure 41: 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.

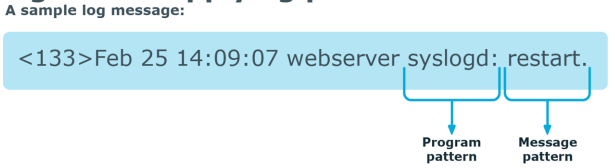
- Rules consist of a message pattern and a class. The Message Pattern is similar to the Program Pattern, but is applied to the message part of the log message (the content of the ${MESSAGE} macro). If a message pattern matches the message, the class of the rule is assigned to the message (for example, 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. This information is displayed by the syslog-ng Premium Edition in the email alerts (if alerts are requested for the rule), and are also displayed on the search interface.
- 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 42: Applying patterns



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

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

NOTE: Wildcards and regular expressions cannot be used in patterns. The @ character must be escaped, 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. The syslog-ng application has a strong preference for rules that match the input string completely.

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 used 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 concerning. However, new messages that have not appeared in the logs before can signal 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 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: 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: 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)
       );
 };
```

Note that if you chain pattern databases, that is, use multiple databases in the same log path, the class assigned to the message (the value of `${.classifier.class}`) will be the one assigned by the last pattern database. As a result, a message might be classified as unknown even if a previous parser successfully classified it. For example, consider the following configuration:

```
log  {
       ...
       parser(db_parser1);
       parser(db_parser2);
       ...
 };
```

Even if `db_parser1` matches the message, `db_parser2` might set `${.classifier.class}` to unknown. To avoid this problem, you can use an 'if' statement to apply the second parser only if the first parser could not classify the message:

```
log  {
       ...
       parser{  db-parser(file("db_parser1.xml"));  };
       if  (match("^unknown$"  value(".classifier.class")))  {
             parser  {  db-parser(file("db_parser2.xml"));  };
       };
       ...
 };
```

For details on how to create your own pattern databases see 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.
- The `.classifier.context_id` macro contains the identifier of the context for messages that were correlated. For details on correlating messages, see Correlating log messages using pattern databases.

Example: 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: 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, One Identity has released (and will continue to release) sample databases. You can download sample pattern databases from the One Identity GitHub page (older samples are temporarily available here).

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 One Identity 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 One Identity 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. To do this, open a GitHub issue, or send them to the syslog-ng mailing list.

Correlating log messages using pattern databases

The syslog-ng PE application can correlate log messages identified using pattern databases. Alternatively, you can also correlate log messages using the `grouping-by()` parser. For details, see Correlating messages using the grouping-by() parser.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix email 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 with syslog-ng PE, you can 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 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. If a message is added to a context, syslog-ng PE

automatically adds the identifier of the context to the `.classifier.context_id` macro of the message.

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: 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 occur immediately, because the difference of the two timestamps (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: 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 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 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: 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 to
${DATE} </value>
            </values>
        </message>
    </action>
</actions>
```

If you do not know in which message of the context contains the information you need, you can use the `grep` template function. For details, see grep.

Example: 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})
```

To perform calculations on fields that have numerical values, see Numerical operations.

Triggering actions for identified messages

The syslog-ng PE application can 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. When used together with message correlation, you can also create a new correlation context when a new message is received.

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: 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: 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: 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>
```

Example: Creating a new context from an action

In syslog-ng PE version 3.87 and newer, you can create a new context as an action. For details, see Element: create-context.

The following example creates a new context whenever the rule matches. The new context receives `1000` as ID, and `program` as scope, and the content set in the `<message>` element of the `<create-context>` element.

```
<rule provider='test' id='12' class='violation'>
    <patterns>
       <pattern>simple-message-with-action-to-create-context</pattern>
    </patterns>
    <actions>
       <action trigger='match'>
          <create-context context-id='1000' context-timeout='60' context-
scope='program'>
             <message inherit-properties='context'>
                <values>
                   <value name='MESSAGE'>context message</value>
                </values>
             </message>
          </create-context>
       </action>
    </actions>
 </rule>
```

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}' == 'myhost'">
```

You can use the same operators in the condition that can be used in filters. For details, see 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: 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 email, you have to route the generated messages to an external application using the `program()` destination.

Example: 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 logpath 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 Correlating log messages using pattern databases.

- 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 Referencing earlier messages of the context.

Example: 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 to ${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: 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: Sending alert when a client disappears

The following example shows how to combine various features of syslog-ng PE to send an email 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 email 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: 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: Using the STRING and ESTRING parsers

For example, look at the following message: `user=joe96 group=somegroup`.

- `@STRING:mytext:@` parses only to the first non-alphanumeric character (`=`), parsing only `user`, so the value of the `${mytext}` macro will be `user`
- `@STRING:mytext:=@` parses the equation mark as well, and proceeds to the next non-alphanumeric character (the whitespace), resulting in `user=joe96`
- `@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.

@EMAIL@

This parser matches an email address. The parameter is a set of characters to strip from the beginning and the end of the email address. That way email addresses enclosed between other characters can be matched easily (for example, `<user@example.com>` or `"user@example.com"`. Characters that are valid for a hostname are not stripped from the end of the hostname. This includes a trailing period if present.

For example, the `@EMAIL:email:"[<]>@` parser will match any of the following email addresses: `<user@example.com>`, `[user@example.com]`, `"user@example.com"`, and set the value of the `email` macro to `user@example.com`.

@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@`.)

@HOSTNAME@

Parses a generic hostname. The hostname may contain only alphanumeric characters (A-Z,a-z,0-9), hypen (-), or dot (.).

@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.

@LLADDR@

Parses a Link Layer Address in the `xx:xx:xx:...` form, where each xx is a 2 digit HEX number (an octet). The parameter specifies the maximum number of octets to match and defaults to 20. The MACADDR parser is a special wrapper using the LLADDR parser. For example, the following parser parses maximally 10 octets, and stores the results in the `link-level-address` macro:

```
@LLADDR:link-level-address:10@
```

@MACADDR@

Parses the standard format of a MAC-48 address, consisting of is six groups of two hexadecimal digits, separated by colons. For example, `00:50:fc:e3:cd:37`.

@NLSTRING@

This parser parses everything until the next new-line character (more precisely, until the next Unix-style LF or Windows-style CRLF character). For single-line messages, NLSTRING is equivalent with ANYSTRING. For multi-line messages, NLSTRING parses to the end of the current line, while ANYSTRING parses to the end of the message. Using NLSTRING is useful when parsing multi-line messages, for example, Windows logs. For example, the following pattern parses information from Windows security auditing logs.

```
<pattern>Example-PC\Example: Security Microsoft Windows security auditing.:
[Success Audit] A new process has been created.

    Subject:
    Security ID: @LNSTRING:.winaudit.SubjectUserSid@
    Account Name: @LNSTRING:.winaudit.SubjectUserName@
    Account Domain: @LNSTRING:.winaudit.SubjectDomainName@
    Logon ID: @LNSTRING:.winaudit.SubjectLogonId@

    Process Information:
    New Process ID: @LNSTRING:.winaudit.NewProcessId@
    New Process Name: @LNSTRING:.winaudit.NewProcessName@
    Token Elevation Type: @LNSTRING:.winaudit.TokenElevationType@
    Creator Process ID: @LNSTRING:.winaudit.ProcessId@
    Process Command Line: @LNSTRING:.winaudit.CommandLine@

    Token Elevation Type indicates the type of token that was assigned to the
new process in accordance with User Account Control policy.</pattern>
```

@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.

@PCRE@

Use Perl-Compatible Regular Expressions (as implemented by the PCRE library), after the identification of the potential patterns has happened by the radix implementation.

Syntax: `@PCRE:name:regexp@`

@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.

@SET@

Parse any combination of the specified characters until another character is found. For example, specifying a whitespace character parses any number of whitespaces, and can be used to process paddings (for example, log messages of the Squid application have whitespace padding after the username).

For example, the `@SET:: "@` parser will parse any combination of whitespaces and double-quotes.

Available in syslog-ng PE 3.4 and later.

@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.

What's new in the syslog-ng pattern database format V5

The V5 database format has the following differences compared to the V4 format:

- The <ruleset> element can now store multiple reference URLs using the new <rule_urls> element. For details, see Element: ruleset.

- In an <action>, you can now initialize a new context. As a result, the <message> element is not required. For details, see Element: create-context.
- The `inherit-properties` attribute is deprecated, use the `inherit-mode` attribute instead. For details, see Element: action.

The syslog-ng pattern database format

Pattern databases are XML files that contain rules describing the message patterns. For sample pattern databases, see Downloading sample pattern databases.

The following scheme describes the V5 format of the pattern database. This format is backwards-compatible with the earlier formats.

For a sample database containing only a single pattern, see Example: A 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 The pdbtool manual page.

To automatically create an initial pattern database from an existing log file, use the `pdbtool patternize` command. For details, see The pdbtool manual page.

Example: A 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='5' 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.

Children

- *patterns*
- *rules*
- *actions*
- *tags*
- *description*: OPTIONAL — A description of the ruleset or the application.
- *url*: OPTIONAL — An URL referring to further information about the ruleset or the application.

- *rule_urls*: OPTIONAL — To list multiple URLs referring to further information about the ruleset or the application, enclose the <url> elements into an <urls> element.

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 email server sends messages using different process names, but all of them begin with the `postfix` string.

  You can also use parsers in the program pattern if needed, and use the parsed results later. For example: `<pattern>postfix\@ESTRING:.postfix.component:[@</pattern>`

  > 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>
```

Using parsers in the program pattern:

```
<pattern>postfix\@ESTRING:.postfix.component:[@</pattern>
```

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. Starting with syslog-ng PE version 3.57.0, if a message is added to a context, syslog-ng PE automatically

adds the identifier of the context to the `.classifier.context_id` macro of the message. For details on correlating messages, see Correlating log messages using pattern databases.

> 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 Correlating log messages using pattern databases.
- *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 following values are available:
  - *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 Correlating log messages using pattern databases.

Children

- *patterns*

Example

```
<rule provider='example' id='f57196aa-75fd-11dd-9bba-001e6806451b'
class='violation'>
```

The following example specifies attributes for correlating messages as well. For details on correlating messages, see Correlating log messages using pattern databases.

```
<rule provider='example' 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: `& < > " '`. 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 Using template functions, for examples, see 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 Correlating log messages using pattern databases.
- *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 Triggering actions for identified messages.

Attributes

N/A

Children

- *action*

Example

Example: 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: 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. When using actions together with message-correlation, you can also use the `$(context-length)` macro, which returns the number of messages in the current context. For example, this can be used to determine if the expected number of messages has arrived to the context: `condition='"$(context-length)" >= "5"'`
- *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`) of the pattern database rule expires. This is available only if actions are used together with correlating messages.

Children

- *create-context*
- *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.
    - For details on the message context, see Correlating log messages using pattern databases and Actions and message correlation. For details on triggering messages, see Triggering actions for identified messages

      inherit-mode: This attribute controls which name-value pairs and tags are propagated to the newly generated message.

        - `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.
        - `last-message`: Only the name-value pairs appearing in the last message are copied. If the context contains only a single message, then it is the message that triggered the action.
        - `none`: An empty message is created, without inheriting any tags or name-value pairs.

      This option is available in syslog-ng PE 3.8 and later.
    - *inherit-properties*: This attribute is deprecated. Use the `inherit-mode` attribute instead.

      If set to `TRUE`, the original message that triggered the action is cloned, including its name-value pairs and tags.

      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 Correlating log messages using pattern databases and Actions and message correlation. For details on triggering messages, see Triggering actions for identified messages

      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: 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: 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: create-context

Location

/patterndb/ruleset/actions/action/*create-context*

Description

OPTIONAL — Creates a new correlation context from the current message and its associated context. This can be used to "split" a context.

Available in syslog-ng PE version 3.87 and later.

Attributes

- *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. Starting with syslog-ng PE version 3.57.0, if a message is added to a context, syslog-ng PE automatically adds the identifier of the context to the `.classifier.context_id` macro of the message. For details on correlating messages, see Correlating log messages using pattern databases.

  > 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 Correlating log messages using pattern databases.

- *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 following values are available:
  - *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 Correlating log messages using pattern databases.

Children

- *message*: A container element storing the message that is added to the new context when the action is executed.
  - *inherit-mode*: This attribute controls which name-value pairs and tags are propagated to the newly generated message.
    - `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.
    - `last-message`: Only the name-value pairs appearing in the last message are copied. If the context contains only a single message, then it is the message that triggered the action.
    - `none`: An empty message is created, without inheriting any tags or name-value pairs.

    For details on the message context, see Correlating log messages using pattern databases and Actions and message correlation. For details on triggering messages, see Triggering actions for identified messages

Example

The following example creates a new context whenever the rule matches. The new context receives `1000` as ID, and `program` as scope, and the content set in the `<message>` element of the `<create-context>` element.

```
<rule provider='test' id='12' class='violation'>
    <patterns>
       <pattern>simple-message-with-action-to-create-context</pattern>
    </patterns>
    <actions>
       <action trigger='match'>
          <create-context context-id='1000' context-timeout='60' context-
scope='program'>
             <message inherit-properties='context'>
                <values>
                   <value name='MESSAGE'>context message</value>
                </values>
             </message>
          </create-context>
       </action>
    </actions>
 </rule>
```

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 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>
```

15

Correlating log messages

The syslog-ng PE application can correlate log messages. Alternatively, you can also correlate log messages using pattern databases. For details, see Correlating log messages using pattern databases.

- To group or correlate log messages that match a set of filters, use the `grouping-by` parser. This works similarly to SQL GROUP BY statements. For details, see Correlating messages using the grouping-by() parser.
- You can correlate log messages identified using pattern databases. For details, see Correlating log messages using pattern databases.

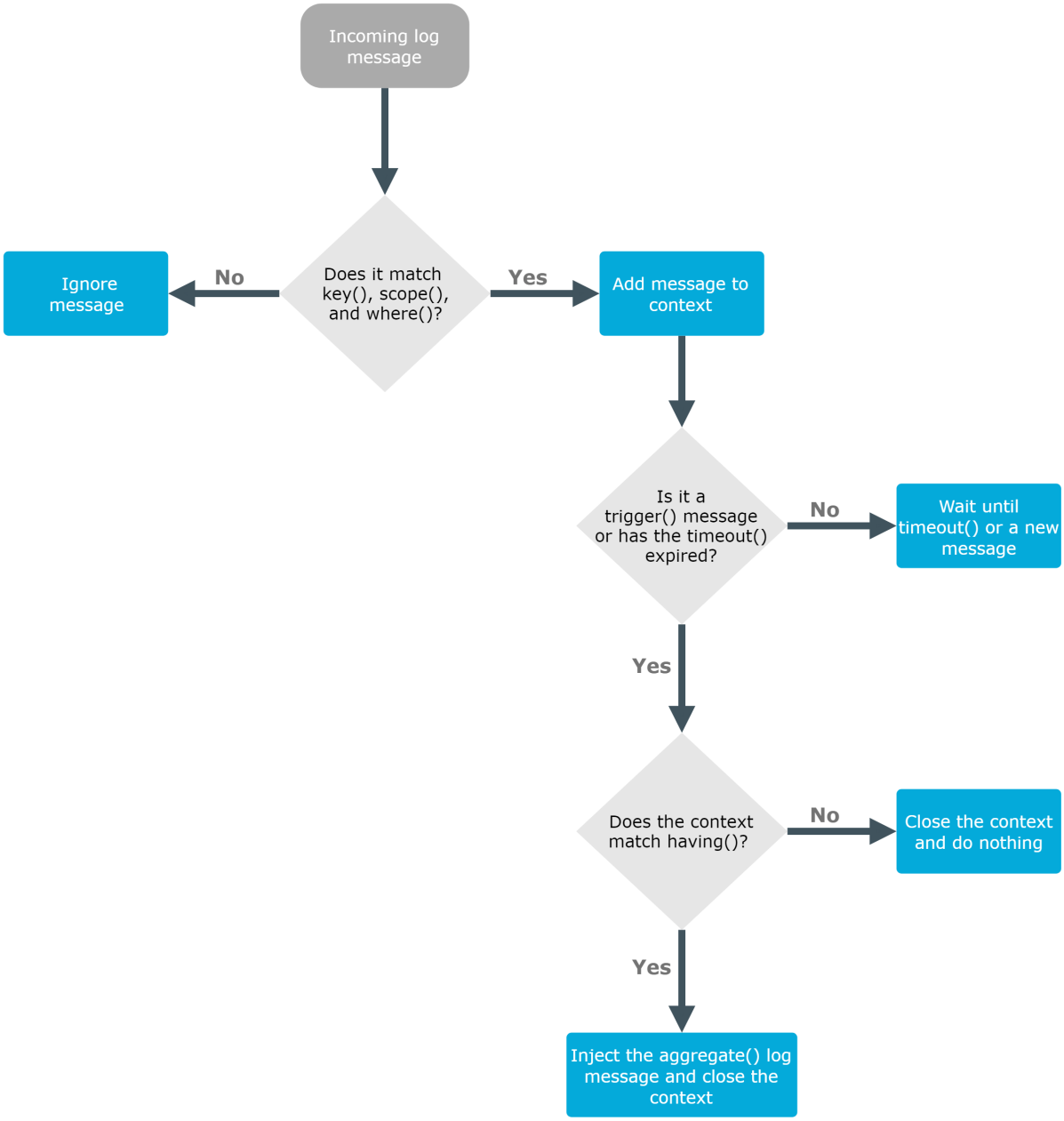
Correlating messages using the `grouping-by()` parser

The syslog-ng PE application can correlate log messages that match a set of filters. This works similarly to SQL GROUP BY statements. Alternatively, you can also correlate log messages using pattern databases. For details, see Correlating log messages 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 email 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 with syslog-ng PE, you can 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.

How the grouping-by() parser works



The `grouping-by()` parser has three options that determine if a message is added to a context: `scope()`, `key()`, and `where()`.

- The `scope()` option acts as an early filter, selecting messages sent by the same process (`${HOST}${PROGRAM}${PID}` is identical), application (`${HOST}${PROGRAM}` is identical), or host.
- The `key()` identifies the context the message belongs to. (The value of the key must be the same for every message of the context.)

- To use a filter to further limit the messages that are added to the context, you can use the `where()` option.

The `timeout()` option determines how long a context is stored, that is, how long syslog-ng PE waits for related messages to arrive. If the group has a specific log message that ends the context (for example, a logout message), you can specify it using the `trigger()` option.

When the context is closed, and the messages match the filter set in the `having()` option (or the `having()` option is not set), syslog-ng PE generates and sends the message set in the `aggregate()` option.

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.

Declaration

```
parser  parser_name  {
       grouping-by(
              key()
              having()
              aggregate()
              timeout()
       );
 };
```

For the parser to work, you must set at least the following options: `key()`, `aggregate()`, and `timeout()`.

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: 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 occur immediately, because the difference of the two timestamps (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: Correlating Linux Audit logs

Linux audit logs tend to be broken into several log messages (generated as a list of lines). Usually, the related lines are close to each other in time, but multiple events can be logged at around the same time, which get mixed up in the output. The example below is the audit log for running `ntpdate`:

```
type=SYSCALL msg=audit(1440927434.124:40347): arch=c000003e syscall=59
success=yes exit=0 a0=7f121cef0b88 a1=7f121cef0c00 a2=7f121e690d98 a3=2
items=2 ppid=4312 pid=4347 auid=4294967295 uid=0 gid=0 euid=0 suid=0
fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="ntpdate"
exe="/usr/sbin/ntpdate" key=(null)
 type=EXECVE msg=audit(1440927434.124:40347): argc=3
a0="/usr/sbin/ntpdate" a1="-s" a2="ntp.ubuntu.com"
 type=CWD msg=audit(1440927434.124:40347):  cwd="/"
 type=PATH msg=audit(1440927434.124:40347): item=0
name="/usr/sbin/ntpdate" inode=2006003 dev=08:01 mode=0100755 ouid=0
ogid=0 rdev=00:00 nametype=NORMAL
 type=PATH msg=audit(1440927434.124:40347): item=1 name="/lib64/ld-linux-
x86-64.so.2" inode=5243184 dev=08:01 mode=0100755 ouid=0 ogid=0
rdev=00:00 nametype=NORMAL
 type=PROCTITLE msg=audit(1440927434.124:40347):
proctitle=2F62696E2F7368002F7573722F7362696E2F6E7470646174652D64656269616E
002D73
```

These lines are connected by their second field: `msg=audit(1440927434.124:40347)`. You can parse such messages using the Linux Audit Parser of syslog-ng PE, and then use the parsed `.auditd.msg` field to group the messages.

```
parser  auditd_groupingby  {
       grouping-by(
             key(".auditd.msg")
             aggregate(
                   value("MESSAGE"  "$(format-json  .auditd.*)")
                   )
             timeout(10)
       );
 };
```

Referencing earlier messages of the context

When creating the aggregated message, or in the various parameters of the `grouping-by()` parser, 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, 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: `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: 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):

```
aggregate(
       value('value  name="MESSAGE"  An  SSH  session  for  ${SSH_USERNAME}@1
from  ${SSH_CLIENT_ADDRESS}@2  closed.  Session  lasted  from  ${DATE}@2  to
${DATE}')
 )
```

If you do not know which message of the context contains the information you need, you can use the `grep` template function. For details, see grep.

Example: 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})
```

To perform calculations on fields that have numerical values, see Numerical operations.

Options of `grouping-by` parsers

The `grouping-by` has the following options.

aggregate()

Synopsis: aggregate()

Description: Specifies the message that syslog-ng PE generates when the context is closed. This option is mandatory.

Note that the `aggregate()` option has access to every message of the context, and has the following options:

- *inherit-mode*: This attribute controls which name-value pairs and tags are propagated to the newly generated message.
  - `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.
  - `last-message`: Only the name-value pairs appearing in the last message are copied. If the context contains only a single message, then it is the message that triggered the action.
  - `none`: An empty message is created, without inheriting any tags or name-value pairs.

  The default value of `inherit-mode()` is `context`.

  For details on the message context, see Correlating messages using the grouping-by() parser.
- *tags*: Adds the specified tag to the list of tags.

- *value*: Adds a name-value pair to the generated message. You can include text, macros, template functions, and you can also reference every message of the context. For details on accessing other messages of the context, see Referencing earlier messages of the context.

having()

Synopsis: having()

Description: Specifies a filter: syslog-ng PE generates the aggregate message only if the result of the filter expression is true. Note that the `having()` filter has access to every message of the context. For details on accessing other messages of the context, see Referencing earlier messages of the context.

inject-mode()

Synopsis: inject-mode()

Description: By default, the aggregated message that syslog-ng PE generates is injected into the same place where the `grouping-by()` 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: Sending triggered messages to the `internal()` source

To send the generated messages to the `internal` source, use the `inject-mode ("internal")` option:

```
parser  p_grouping-by  {grouping-by(
       ...
       inject-mode("internal")
 );};
```

To inject the generated messages where the parser is referenced, use the `inject-mode("pass-through")` option:

```
parser  p_grouping-by  {grouping-by(
       ...
       inject-mode("pass-through")
 );};
```

You can configure the generated message in the `aggregate()` option (see aggregate()). You can create an entire message, use macros and values extracted from the original message, and so on.

key()

Synopsis: key()

Description: Specifies the key (that is, the name of a name-value pair) that every message must have in order to be added to the context. The value of the key must be the same for every message of the context. For example, this can be a session-id parsed from firewall messages, and so on.

This is a mandatory option.

NOTE: Messages that do not have a key will all belong to the same context.

scope()

Synopsis: scope()

Description: Specifies which messages belong to the same context. The following values are available:

- *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.
- *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. This is the default value.

timeout()

Synopsis: timeout([seconds])

Description: Specifies the maximum time to wait for all messages of the context to arrive. If no new message is added to the context during this period, the context is assumed to be complete and syslog-ng PE generates and sends the triggered message (specified in the aggregate() option), and clears the context. If a new message is added to the context, the timeout period is restarted.

This option is mandatory, and its value must be equal to or greater than 1.

trigger()

Synopsis: trigger()

Description: A filter that specifies the final message of the context. If the filter matches the incoming message, syslog-ng PE generates and sends the triggered message (specified in the aggregate() option), and clears the context.

where()

Synopsis: where()

Description: Specifies a filter condition. Messages not matching the filter will not be added to the context. Note that the `where()` filter has access only to the current message.

16

Enriching log messages with external data

To properly interpret the events that the log messages describe, you must be able to handle log messages as part of a system of events, instead of individual information chunks. The syslog-ng PE application allows you to import data from external sources to include in the log messages, thus extending, enriching, and complementing the data found in the log message.

The syslog-ng PE application currently provides the following possibilities to enrich log messages.

- You can add name-value pairs from an external CSV file. For details, see Adding metadata from an external file.
- You can resolve the IP addresses from log messages to include GeoIP information in the log messages. For details, see Looking up GeoIP2 data from IP addresses.
- You can write custom Python modules to process the messages and add data from external files or databases. For details, see Python parser.

Adding metadata from an external file

In syslog-ng PE version 3.87.0 and later, you can use an external database file to add additional metadata to your log messages. For example, you can create a database (or export it from an existing tool) that contains a list of hostnames or IP addresses, and the department of your organization that the host belongs to, the role of the host (mailserver, webserver, and so on), or similar contextual information.

The database file is a simple text file in comma-separated value (CSV) format, where each line contains the following information:

- A selector or ID that appears in the log messages, or the name of a filter that matches the messages, for example, the hostname.
- The name of the name-value pair that syslog-ng PE adds to matching log messages.
- The value of the name-value pairs.

For example, the following csv-file contains three lines identified with the IP address, and adds the `host-role` field to the log message.

```
192.168.1.1,host-role,webserver
192.168.2.1,host-role,firewall
192.168.3.1,host-role,mailserver
```

The database file

The database file must comply with the RFC4180 CSV format, with the following exceptions and limitations:

- The values of the CSV-file cannot contain line-breaks

To add multiple name-value pairs to a message, include a separate line in the database for each name-value pair, for example:

```
192.168.1.1,host-role,webserver
192.168.1.1,contact-person,"John Doe"
192.168.1.1,contact-email,johndoe@example.com
```

Technically, `add-contextual-data()` is a parser in syslog-ng PE so you have to define it as a parser object.

Declaration

```
parser  p_add_context_data  {
       add-contextual-data(
             selector("$HOST"),
             database("context-info-db.csv"),
       );
 };
```

You can also add data to messages that do not have a matching selector entry in the database using the `default-selector()` option.

If you modify the database file, you have to reload syslog-ng PE for the changes to take effect. If reloading syslog-ng PE or the database file fails for some reason, syslog-ng PE will keep using the last working database file.

Example: Adding metadata from a CSV file

The following example defines uses a CSV database to add the role of the host based on its IP address, and prefixes the added name-value pairs with `.metadata`. The destination includes a template that simply appends the added name-value pairs to the end of the log message.

```
@include "scl.conf"

source s_network {
      network(port(5555));
};

destination d_local {
      file("/tmp/test-msgs.log"
      template("$MSG Additional metadata:[${.metadata.host-role}]")};

parser p_add_context_data {
      add-contextual-data(
            selector("$SOURCEIP"),
            database("context-info-db.csv"),
            default-selector("unknown"),
            prefix(".metadata.")
      );
};

log {
      source(s_network);
      parser(p_add_context_data);
      destination(d_local);
};
```

```
192.168.1.1,host-role,webserver
192.168.2.1,host-role,firewall
192.168.3.1,host-role,mailserver
unknown,host-role,unknown
```

Using filters as selector

To better control to which log messages you add contextual data, you can use filters as selectors. In this case, the first column of the CSV database file must contain the name of a filter. For each message, syslog-ng PE evaluates the filters in the order they appear in the database file. If a filter matches the message, syslog-ng PE adds the name-value pair related to the filter.

For example, the database file can contain the entries. (For details on the accepted CSV-format, see database().)

```
f_auth,domain,all
f_localhost,source,localhost
f_kern,domain,kernel
```

Note that syslog-ng PE does not evaluate other filters after the first match. For example, if you use the previous database file, and a message matches both the `f_auth` and `f_localhost` filters, syslog-ng PE adds only the name-value pair of `f_auth` to the message.

To add multiple name-value pairs to a message, include a separate line in the database for each name-value pair, for example:

```
f_localhost,host-role,firewall
f_localhost,contact-person,"John Doe"
f_localhost,contact-email,johndoe@example.com
```

You can also add data to messages that do not have a matching selector entry in the database using the `default-selector()` option.

You must store the filters you reference in a database in a separate file. This file is similar to a syslog-ng PE configuration file, but must contain only a version string and filters (and optionally comments). You can use the `syslog-ng --syntax-only <filename>` command to ensure that the file is valid. For example, the content of such a file can be:

```
@version: 7.0
filter f_localhost { host("mymachine.example.com") };
filter f_auth { facility(4) };
filter f_kern { facility(0) };
```

Declaration

```
parser  p_add_context_data_filter  {
       add-contextual-data(
             selector(filters("filters.conf")),
             database("context-info-db.csv"),
             prefix(".metadata.")
       );
 };
```

If you modify the database file, or the file that contains the filters, you have to reload syslog-ng PE for the changes to take effect. If reloading syslog-ng PE or the files fails for some reason, syslog-ng PE will keep using the last working version of the file.

Options add-contextual-data()

The `add-contextual-data()` has the following options.

Required options

The following options are required: `selector()`, `database()`.

database()

Type:	<path-to-file>.csv
Default:	

Description: Specifies the path to the CSV file, for example, `/opt/syslog-ng/my-csv-database.csv`. The extension of the file must be `.csv`, and can include Windows-style (CRLF) or UNIX-style (LF) linebreaks. You can use absolute path, or relative to the syslog-ng PE binary.

default-selector()

Synopsis:	default-selector()

Description: Specifies the ID of the entry (line) that is corresponds to log messages that do not have a selector that matches an entry in the database. For example, if you add name-value pairs from the database based on the hostname from the log message (`selector("${HOST}")`), then you can include a line for unknown hosts in the database, and set `default-selector()` to the ID of the line for unknown hosts. In the CSV file:

```
unknown-hostname,host-role,unknown
```

In the syslog-ng PE configuration file:

```
add-contextual-data(
       selector("$HOST")
       database("context-info-db.csv")
       default-selector("unknown-hostname")
 );
```

prefix()

Synopsis:	prefix()

Description: Insert a prefix before the name part of the added name-value pairs (including the pairs added by the `default-selector()`) to help further processing.

selector()

Synopsis:	selector()

Description: Specifies the string or macro that syslog-ng PE evaluates for each message, and if its value matches the ID of an entry in the database, syslog-ng PE adds the name-value pair of every matching database entry to the log message. Currently, you can use

strings and a single macro (for example, `${HOST}`) in the `selector()` option, templates are not supported. To use filters as selectors, see Using filters as selector.

Looking up GeoIP2 data from IP addresses

The syslog-ng PE application can lookup IP addresses from an offline GeoIP2 database, and make the retrieved data available in name-value pairs. Depending on the database used, you can access country code, longitude, and latitude information and so on.

The syslog-ng PE application works with the Country and the City version of the GeoIP2 database, both free and the commercial editions. The syslog-ng PE application works with the `mmdb` (GeoIP2) format of these databases. Other formats, like `csv` are not supported.

NOTE: To access longitude and latitude information, download the City version of the GeoIP2 database.

There are two types of GeoIP2 databases available.

- *GeoLite2 City:*
  - free of charge
  - less accurate
- *GeoIP2 City:*
  - has to be purchased
  - more accurate

Unzip the downloaded database (for example, to the `/usr/share/GeoIP2/GeoIP2City.mmdb` file). This path will be used later in the configuration.

Starting with version 3.247.0.17, syslog-ng PE tries to automatically detect the location of the database. If that is successful, the `database()` option is not mandatory.

Options of `geoip2` parsers

The `geoip2` 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 Hard versus 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 `.geoip2` prefix, use the `prefix(.geoip2)` option. To refer to a particular data when using a prefix, use the prefix in the name of the macro, for example, `${geoip2.country_code}` .

database()

Synopsis:	database()
Default:	

Description: Path to the GeoIP2 database to use. This works with absolute and relative paths as well. Note that syslog-ng PE must have the required privileges to read this file. Do not modify or delete this file while syslog-ng PE is running, it can crash syslog-ng PE.

Starting with version 3.247.0.17, syslog-ng PE tries to automatically detect the location of the database. If that is successful, the `database()` option is not mandatory.

Monitoring statistics and metrics of syslog-ng

The syslog-ng PE application collects various statistics and measures different metrics about the messages it receives and delivers. These metrics are collected into different counters, depending on the configuration of syslog-ng PE. The `stats-level()` global option determines exactly which statistics syslog-ng PE collects. You can access these statistics and metrics using the following methods.

Recommended: Structured, selective methods:

- Using the `monitoring()` source.
- Using the `syslog-ng-ctl` query command.

  For further information about using `syslog-ng-ctl` commands, see The syslog-ng manual pages.

Legacy: Unstructured, bulk methods:

- Using the `internal()` source.
- Using the `syslog-ng-ctl stats` command.

  For further information about using `syslog-ng-ctl` commands, see The syslog-ng manual pages.
- 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.

Metrics and counters of syslog-ng PE

You can list all active metrics on your syslog-ng PE host using the following command (this lists the metrics, without their current values): `syslog-ng-ctl query list "*"`

To list the metrics and their values, use the following command: `syslog-ng-ctl query get "*"`

The displayed metrics have the following structure.

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. These counters only appear above a certain value of `stats-level()` global option:
        - `host`: source host, from `stats-level(2)`
        - `program`: program, from `stats-level(3)`
        - `sender`: sender host, from `stats-level(3)`

Example: Dynamic counters

The following example contains 6 different dynamic values: a sender, a host, and four different programs.

```
src.sender;;localhost;d;processed;4
src.sender;;localhost;d;stamp;1509121934
src.program;;P-18069;d;processed;1
src.program;;P-18069;d;stamp;1509121933
src.program;;P-21491;d;processed;1
src.program;;P-21491;d;stamp;1509121934
src.program;;P-9774;d;processed;1
src.program;;P-9774;d;stamp;1509121919
src.program;;P-14737;d;processed;1
src.program;;P-14737;d;stamp;1509121931
src.host;;localhost;d;processed;4
src.host;;localhost;d;stamp;1509121934
```

To avoid performance issues or even overloading syslog-ng PE, you might want to limit the number of registered dynamic counters in the message statistics. To do this, configure the stats-max-dynamics() global option.

- 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 `connections` statistics counter displays the number of connections tracked by syslog-ng PE for the selected source driver.

Example: sample configuration and statistics output

The following configuration will display the following `syslog-ng-ctl` statistics output:

Configuration:

```
source  s_network  {
    tcp(
       port(8001)
    );
};
```

Statistics output:

```
src.tcp;s_network#0;tcp,127.0.0.5;a;processed;1
src.tcp;s_network#0;tcp,127.0.0.1;a;processed;3
src.tcp;s_network;afsocket_sd.(stream,AF_INET
(0.0.0.0:8001));a;connections;2
```

6. The type of the statistics:
   - `batch_size_avg`: When batching is enabled, then this shows the current average batch size of the given source or destination.

     NOTE: In version 7.0.27, syslog-ng PE only supports the `batch_size_avg` for the `http()` destination.
   - `batch_size_max`: When batching is enabled, the value of `batch_size_max` shows the current largest batch size of the given source or destination.

     NOTE: In version 7.0.27, syslog-ng PE only supports the `batch_size_max` for the `http()` destination.
   - `discarded`: The number of messages discarded by the given parser. These are messages that the parser could not parsed, and are therefore not processed. For example:

     ```
     parser;demo_parser;;a;discarded;20
     ```

- `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, or syslog-ng PE dropped the message for some other reason (for example, a parsing error).
- `eps_last_1h`: The EPS value of the past 1 hour.
- `eps_last_24h`: The EPS value of the past 24 hours.
- `eps_since_start`: The EPS value since the current syslog-ng PE start.

  NOTE: When using the `eps_last_1h`, the `eps_last_24h`, and the `eps_since_start` statistics, consider the following:

  - EPS stands for "event per second", and in our case, a message received or sent counts as a single event.
  - The `eps_last_1h`, the `eps_last_24h`, and the `eps_since_start` values are only approximate values.
  - The `eps_last_1h`, the `eps_last_24h`, and the `eps_since_start` values are automatically updated every `60` seconds.
- `matched`: The number of messages that are accepted by a given filter. Available for filters and similar objects (for example, a conditional rewrite rule). For example, if a filter matches a specific hostname, then the `matched` counter contains the number of messages that reached the filter from this hosts.

  ```
  filter;demo_filter;;a;matched;28
  ```
- `memory_usage`: The memory used by the messages in the different queue types (in bytes). This includes every queue used by the object, including memory buffers (log-fifo) and disk-based buffers (both reliable and non-reliable). For example:

  ```
  dst.network;d_net#0;tcp,127.0.0.1:9999;a;memory_usage;0
  ```

  NOTE: The memory usage (size) of queues is not equal to the memory usage (size) of the log messages in syslog-ng PE. A log message can be in multiple queues, thus its size is added to multiple queue sizes. To check the size of all log messages, use `global.msg_allocated_bytes.value` metric.
- `msg_size_max`: The current largest message size of the given source or destination.
- `msg_size_avg`: The current average message size of the given source or destination.

  NOTE: When using the `msg_size_avg` and `msg_size_max` statistics, consider that message sizes are calculated as follows:

  - on the source side: the length of the incoming raw message
  - on the destination side: the length of the outgoing formatted message

- `not_matched`: The number of messages that are filtered out by a given filter. Available for filters and similar objects (for example, a conditional rewrite rule). For example, if a filter matches a specific hostname, then the `not_matched` counter contains the number of messages that reached the filter from other hosts, and so the filter discarded them.

  NOTE: Since the `not_matched` metric applies to filters, and filters are expected to discard messages that do not match the filter condition, `not_matched` messages are not included in the `dropped` metric of other objects.

  ```
  filter;demo_filter;;a;not_matched;0
  ```

- `processed`: The number of messages that successfully reached their destination driver.

  NOTE: Consider that a message that has successfully reached its destination driver does not necessarily mean that the destination driver successfully delivered the messages as well. For example, a message can be written to disk or sent to a remote server after reaching the destination driver.

- `queued`: The number of messages passed to the message queue of the destination driver, waiting to be sent to the destination.
- `stamp`: The UNIX timestamp of the last message sent to the destination.
- `suppressed`: The number of suppressed messages (if the `suppress()` feature is enabled).
- `written`: The number of messages successfully delivered to the destination. This value is calculated from other counters: `written = processed - queued - dropped`. That is, the number of messages syslog-ng PE passed to the destination driver (processed) minus the number of messages that are still in the output queue of the destination driver (queued) and the number of messages dropped because of an error (dropped, for example, because syslog-ng PE could not deliver the message to the destination and exceeded the number of retries).

  This metric is calculated from other metrics. You cannot reset this metric directly: to reset it, you have to reset the metrics it is calculated from.

NOTE: Consider that for syslog-ng PE version 7.0.27, the following statistics counters are only supported for the `http()` destination, or the `http()` destination and all `network()` sources and destinations, and all `file()` sources and destinations, respectively:

- `msg_size_max`
- `msg_size_avg`
- `batch_size_max`
- `batch_size_avg`
- `eps_last_1h`

- `eps_last_24h`
- `eps_since_start`

7. The number of such messages.

Availability of statistics

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

- 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.

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.

Aggregated statistics

Aggregated statistics are available for different sources and destinations from different levels and upwards:

	msg_ size_ avg	**msg_ size_ max**	**batch_ size_avg**	**batch_ size_max**	**eps_ last_ 1h**	**eps_ last_ 1h**	**eps_ last_ 1h**
`network()` source and destination	from level 1	from level 1	counter N/A	counter N/A	from level 1	from level 1	from level 1
`file()` source and destination	from level 1	from level 1	counter N/A	counter N/A	from level 1	from level 1	from level 1
`http()` destination	from level 0	from level 0	from level 0	from level 0	from level 0	from level 0	from level 0

Log statistics from the `internal()` source

If the `stats-freq()` global option is higher than 0, syslog-ng PE periodically sends a log statistics message. This message contains 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'
```

The statistics include a list of source groups and destinations, as well as the number of processed messages for each. You can control the verbosity of the statistics using the `stats-level()` global option. The following is an example output.

```
src.internal;s_all#0;;a;processed;6445
src.internal;s_all#0;;a;stamp;1268989330
destination;df_auth;;a;processed;404
destination;df_news_dot_notice;;a;processed;0
destination;df_news_dot_err;;a;processed;0
destination;d_ssb;;a;processed;7128
destination;df_uucp;;a;processed;0
source;s_all;;a;processed;7128
destination;df_mail;;a;processed;0
destination;df_user;;a;processed;1
destination;df_daemon;;a;processed;1
destination;df_debug;;a;processed;15
destination;df_messages;;a;processed;54
destination;dp_xconsole;;a;processed;671
dst.tcp;d_network#0;10.50.0.111:514;a;dropped;5080
dst.tcp;d_network#0;10.50.0.111:514;a;processed;7128
dst.tcp;d_network#0;10.50.0.111:514;a;queued;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:

To reset the statistics to zero, use the following command: `syslog-ng-ctl stats --reset`

The monitoring() source

The monitoring() source allows you to select which statistics of syslog-ng PE you want to monitor. In addition, the statistics are available as structured name-value pairs, so you can format the output similarly to other log messages. That way, you can easily convert the statistics and metrics, for example, into JSON or WELF format. That way, you can send the statistics of your log messages into a monitoring solution, for example, into Riemann, Redis, or Graphite.

The monitoring() source queries the statistics (counters) that syslog-ng PE collects, formats them, and optionally resets the counters. The monitoring() source emits only these messages, making it easy to route them to their appropriate destination. The stats-level() global option determines exactly which statistics syslog-ng PE collects.

Declaration

```
source  s_monitor{
    monitoring(
       query("*")
    );
 };
```

Example: Save all statistics into a file in JSON format

The following configuration increases the stats-level() option to 3, and generates a JSON-formatted message every 10 seconds. The generated message contains every available statistics, and is saved into the /var/log/syslog-ng-statistics.log file.

```
@version: 7.0
options {
  stats-level(3);
  keep-hostname(no);
};

source s_monitor { monitoring(
    query("*")
    freq(1)
    message-template("$(format-flat-json --scope nv_pairs)")
  );
};

destination d_file {
  file("/var/log/syslog-ng-statistics.log");
```

```
};

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

The generated message is similar to this one:

```
[2021-09-02T13:30:18.003557] Outgoing message; message='Sep 2 13:30:18
test-host syslog-ng[71345]: {"tag..source.s_
monitor.processed":"111","src.severity.7.processed":"0","src.severity.6.pr
ocessed":"111","src.severity.5.processed":"0","src.severity.4.processed":"
0","src.severity.3.processed":"0","src.severity.2.processed":"0","src.seve
rity.1.processed":"0","src.severity.0.processed":"0","src.sender.s_
monitor#0.test-host.stamp":"1630582216","src.sender.s_monitor#0.test-
host.processed":"111","src.sender.test-
host.stamp":"1630582216","src.sender.test-
host.processed":"111","src.program.syslog-
ng.stamp":"1630582216","src.program.syslog-
ng.processed":"111","src.monitoring.s_
monitor#0.stamp":"1630582216","src.monitoring.s_
monitor#0.processed":"111","src.host.s_monitor#0.test-
host.stamp":"1630582216","src.host.s_monitor#0.test-
host.processed":"111","src.host.test-
host.stamp":"1630582216","src.host.test-
host.processed":"111","src.facility.other.pr'
```

For reference, the JSON part in a readable format is:

```
{ "tag..source.s_monitor.processed": "111", "src.severity.7.processed":
"0", "src.severity.6.processed": "111", "src.severity.5.processed": "0",
"src.severity.4.processed": "0", "src.severity.3.processed": "0",
"src.severity.2.processed": "0", "src.severity.1.processed": "0",
"src.severity.0.processed": "0", "src.sender.s_monitor#0.test-host.stamp":
"1630582216", "src.sender.s_monitor#0.test-host.processed": "111",
"src.sender.test-host.stamp": "1630582216", "src.sender.test-
host.processed": "111", "src.program.syslog-ng.stamp": "1630582216",
"src.program.syslog-ng.processed": "111", "src.monitoring.s_
monitor#0.stamp": "1630582216", "src.monitoring.s_monitor#0.processed":
"111", "src.host.s_monitor#0.test-host.stamp": "1630582216", "src.host.s_
monitor#0.test-host.processed": "111", "src.host.test-host.stamp":
"1630582216", "src.host.test-host.processed": "111",
"src.facility.other.processed": "0", "src.facility.9.processed": "0",
"src.facility.8.processed": "0", "src.facility.7.processed": "0",
```

```
"src.facility.6.processed": "0", "src.facility.5.processed": "111",
"src.facility.4.processed": "0", "src.facility.3.processed": "0",
"src.facility.23.processed": "0", "src.facility.22.processed": "0",
"src.facility.21.processed": "0", "src.facility.20.processed": "0",
"src.facility.2.processed": "0", "src.facility.19.processed": "0",
"src.facility.18.processed": "0", "src.facility.17.processed": "0",
"src.facility.16.processed": "0", "src.facility.15.processed": "0",
"src.facility.14.processed": "0", "src.facility.13.processed": "0",
"src.facility.12.processed": "0", "src.facility.11.processed": "0",
"src.facility.10.processed": "0", "src.facility.1.processed": "0",
"src.facility.0.processed": "0", "source.s_monitor.processed": "111",
"global.sdata_updates.processed": "0", "global.scratch_buffers_
count.queued": "280", "global.scratch_buffers_bytes.queued": "50944",
"global.payload_reallocs.processed": "557", "global.msg_clones.processed":
"0", "global.msg_allocated_bytes.value": "1056", "dst.file.d_
file#0./var/log/syslog-ng-statistics.log.written": "111", "dst.file.d_
file#0./var/log/syslog-ng-statistics.log.truncated_count": "0",
"dst.file.d_file#0./var/log/syslog-ng-statistics.log.truncated_bytes":
"0", "dst.file.d_file#0./var/log/syslog-ng-statistics.log.queued": "0",
"dst.file.d_file#0./var/log/syslog-ng-statistics.log.processed": "111",
"dst.file.d_file#0./var/log/syslog-ng-statistics.log.msg_size_max":
"3019", "dst.file.d_file#0./var/log/syslog-ng-statistics.log.msg_size_
avg": "2991", "dst.file.d_file#0./var/log/syslog-ng-statistics.log.memory_
usage": "0", "dst.file.d_file#0./var/log/syslog-ng-statistics.log.eps_
since_start": "1", "dst.file.d_file#0./var/log/syslog-ng-
statistics.log.eps_last_24h": "1", "dst.file.d_file#0./var/log/syslog-ng-
statistics.log.eps_last_1h": "1", "dst.file.d_file#0./var/log/syslog-ng-
statistics.log.dropped": "0", "destination.d_file.processed": "111",
"center.received.processed": "111", "center.queued.processed": "111",
"PROGRAM": "syslog-ng", "PID": "71345" }
```


`monitoring()` source options

The `monitoring()` driver has the following options. Only the `query()` option is required, other options are optional.

clear-on-read()

Type:	boolean
Default:	no

Description: Reset the counters after reading. Note that if a destination is not available, syslog-ng PE will not reset its counter even if `clear-on-read()` is set to `yes`.

If you use multiple monitoring source, and you use the `clear-on-read()` parameter, make sure to adjust the queries appropriately. Overlapping queries that read and reset the same counters result in incorrect statistics.

freq()

Type:	integer
Default:	600 [seconds]

Description: Specifies how often does syslog-ng PE execute the query and send a statistics message.

message-template()

Type:	string
Default:	N/A

Description: Specifies how the message containing the queried statistics is formatted. You can use macros and template functions in the format string. For example, you can format the message as a JSON object:

```
source s_monitor{ monitoring(
       query("*")
       freq(10)
       message-template('$(format-json --scope nv_pairs)')
       );};
```

Note that here you can only format the payload of the message (that is the, ${MESSAGE} part). You can format the headers or other parts of the outgoing message in the destination driver.

query()

Type:	string
Default:	N/A

Description: Specifies which statistical counters will be included in the messages. Note that the list of available counters depends on your syslog-ng PE configuration (mainly the configured sources and destinations) and on the `stats-level()` global option. The `*` string includes every available counters. The syntax of the `query` option is identical to the `syslog-ng-ctl query get <query>` command.

```
source  s_monitor{
    monitoring(
       query("*")
    );
 };
```

For example, the `"destination*"` query lists the configured destinations, and the metrics related to each destination. An example output:

```
destination.d_file.processed=111
```

The `monitoring-welf()` source

This source is actually preconfigured `monitoring()` source that generates statistics messages in WELF format. Starting with syslog-ng PE version 7.0.2, syslog-ng PE uses this driver for new installations to generate statistics. By default, a message is sent every 10 minutes (600 seconds).

```
@version:  7.0
@include  'scl.conf'
options  {
      stats_level(3);
};

source  s_monitoring_welf  {
      monitoring-welf(freq(10)  query('*'));
};

destination  d_file  {
      file("/tmp/output.txt");
};

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

The output is similar to the following:

```
[2021-09-02T14:05:16.520260] Outgoing message; message='Sep 2 14:05:16 test-host
syslog-ng[73205]: PID=73205 PROGRAM=syslog-ng center.queued.processed=3
center.received.processed=3 destination.d_file.processed=3 dst.file.d_
file#0./tmp/output.txt.dropped=0 dst.file.d_file#0./tmp/output.txt.eps_last_1h=0
dst.file.d_file#0./tmp/output.txt.eps_last_24h=0 dst.file.d_
file#0./tmp/output.txt.eps_since_start=0 dst.file.d_
file#0./tmp/output.txt.memory_usage=0 dst.file.d_file#0./tmp/output.txt.msg_
```

```
size_avg=2221 dst.file.d_file#0./tmp/output.txt.msg_size_max=2602 dst.file.d_
file#0./tmp/output.txt.processed=3 dst.file.d_file#0./tmp/output.txt.queued=0
dst.file.d_file#0./tmp/output.txt.truncated_bytes=0 dst.file.d_
file#0./tmp/output.txt.truncated_count=0 dst.file.d_
file#0./tmp/output.txt.written=3 global.msg_allocated_bytes.value=1056
global.msg_clones.processed=0 global.payload_reallocs.processed=15
global.scratch_buffers_bytes.queued=24576 global.scratch_buffers_
count.queued=140 global.sdata_updates.processed=0 source.s_monitoring_
welf.processed=3 src.facility'
```

```
Apr 3 14:03:26 example-host syslog-ng[12363]: PID=12363 PROGRAM=syslog-ng
center.queued.processed=0 center.received.processed=0 destination.d_
file.processed=0 global.msg_clones.processed=0 global.payload_
reallocs.processed=2 global.sdata_updates.processed=0 source.s_monitoring_
welf.processed=0 src.facility.0.processed=0 src.facility.1.processed=0
src.facility.10.processed=0 src.facility.11.processed=0
src.facility.12.processed=0 src.facility.13.processed=0
src.facility.14.processed=0 src.facility.15.processed=0
src.facility.16.processed=0 src.facility.17.processed=0
src.facility.18.processed=0 src.facility.19.processed=0
src.facility.2.processed=0 src.facility.20.processed=0
src.facility.21.processed=0 src.facility.22.processed=0
src.facility.23.processed=0 src.facility.3.processed=0
src.facility.4.processed=0 src.facility.5.processed=0 src.facility.6.processed=0
src.facility.7.processed=0 src.facility.8.processed=0 src.facility.9.processed=0
src.facility.other.processed=0 src.monitoring.s_monitoring_welf#0.processed=0
src.monitoring.s_monitoring_welf#0.stamp=0 src.severity.0.processed=0
src.severity.1.processed=0 src.severity.2.processed=0 src.severity.3.processed=0
src.severity.4.processed=0 src.severity.5.processed=0 src.severity.6.processed=0
src.severity.7.processed=0\x0a'
```

18

Multithreading and scaling in syslog-ng PE

Starting with version 4 F13.3, syslog-ng PE can process sources and destinations in multithreaded mode to scale to multiple CPUs or cores for increased performance. Starting with version 3.65 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 7.

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 The syslog-ng manual page), 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 `udp-balancer` source uses a thread for each listener configured in its `listeners()` option.
- 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 3.65 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: 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

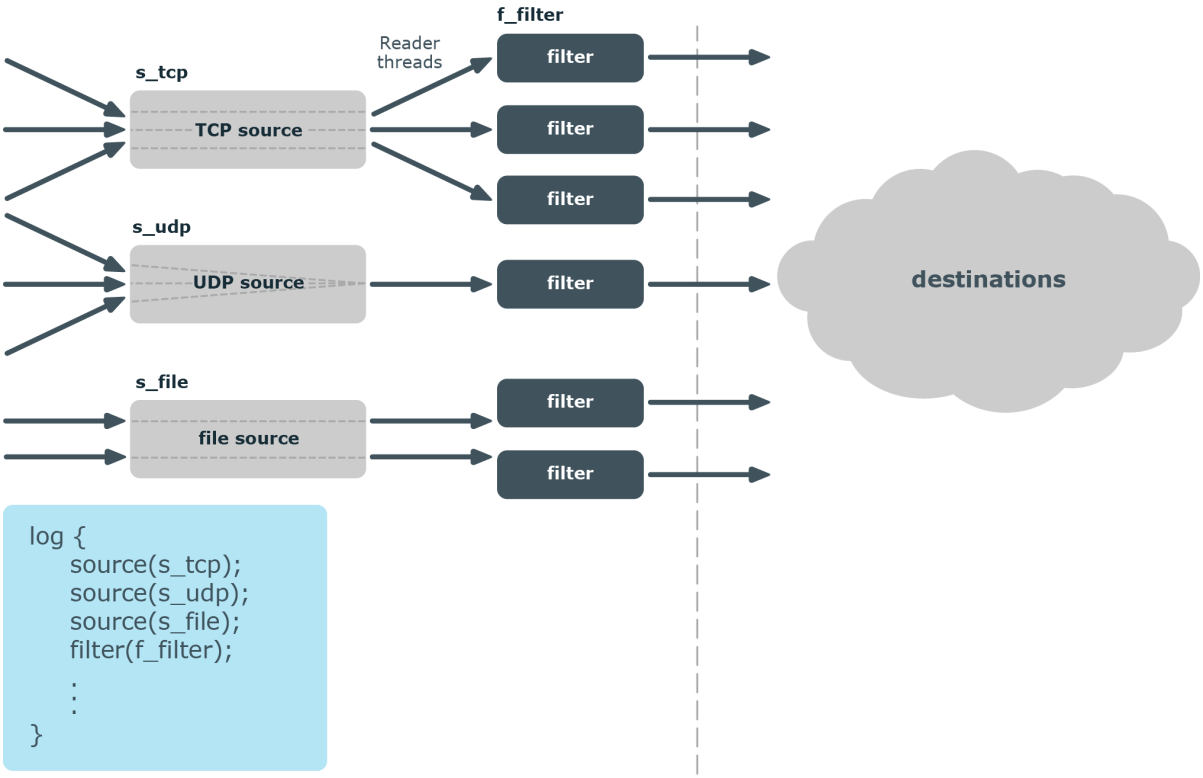
File sources scale based on the number of files that the syslog-ng PE is reading. If there are 10 files all coming to the same source, then that source can use 10 threads, one thread for each file.

NOTE: When collecting log messages from multiple files, the file source is a `wildcard-file()` source.

TCP-based network sources scale based on the number of active connections. This means that if there are 10 incoming connections all coming to the same 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 43: 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.

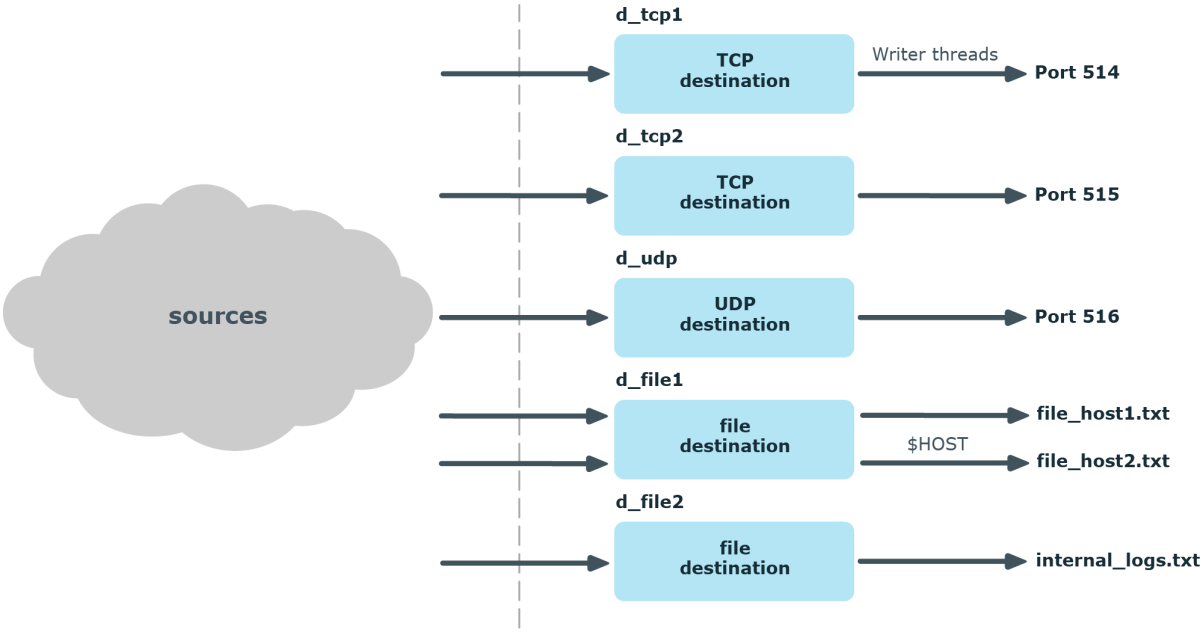
Destinations

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 44: How multithreading works — destinations



Troubleshooting syslog-ng

This chapter provides tips and guidelines about troubleshooting problems related to syslog-ng.

- 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 persists, 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 may 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 The syslog-ng control tool manual page.
- 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 may seem to work, but result in losing parts of the messages. For details on the working combinations, see 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: If your syslog-ng PE host uses an NFS partition, see Using syslog-ng PE with NFS or CIFS (or SMB) 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 Configuring flow-control). For more information about the error caused by the missing flow-control, see Destination queue full.

  The number of dropped messages is displayed per destination in the log message statistics of syslog-ng (for details, see Monitoring 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 Collecting log messages from UDP sources tutorial.
- 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 Monitoring 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, ALTP).

    To make syslog-ng PE forget old clients that do not exist anymore, enable the `reset-license-counter()` global option.

- 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.

Creating syslog-ng core files

When syslog-ng crashes for some reason, it can create a core file that contains important troubleshooting information.

To enable core files

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.

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 One Identity 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 -d`

TIP: To execute these commands on an already running syslog-ng PE process, use the `-p <pid_of_syslog-ng>` parameter.

Running a failure script

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 email 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

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.

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 The syslog-debun manual page. For support contacts, see About us.

Error messages

This section describes the most common error messages.

Destination queue full

Error message:	`Destination queue full, dropping messages; queue_len='10000', log_fifo_size='10000', count='4', persist_name='afsocket_dd_qfile(stream,serverdown:514)'`
Description:	This message indicates message loss. Flow-control must be enabled in the log path. When flow-control is enabled, syslog-ng will stop reading messages from the sources of the log statement if the destinations are not able to process the messages at the required speed. If flow-control is enabled, syslog-ng will only drop messages if the destination queues/window sizes are improperly sized.
Solution:	Enable flow-control in the log path. If flow-control is disabled, syslog-ng will drop messages if the destination queues are full. Note that syslog-ng will drop messages even if the server is alive. If the remote server accepts logs at a slower rate than the sender syslog-ng receives them, the sender syslog-ng will fill up the destination queue, then drop the newer messages. Sometimes this error occurs only at a specific time interval, for example, only between `7:00` AM and `8:00` AM or between `16:00` PM and `17:00` PM when your users log in or log off and that generates a lot of messages within a short interval.

For more information, see Managing incoming and outgoing messages with flow-control.

Alert unknown CA

Error message:	`SSL error while writing stream; tls_error='SSL routines:ssl3_read_bytes:tlsv1 alert unknown ca'`
Description:	This message indicates that the other (remote) side could not verify the certificate sent by syslog-ng.
Solution:	Check the logs on the remote site and identify why the receiving syslog-ng could not find the CA certificate that signed this certificate.

PEM routines:PEM_read_bio:no start line

Error message:	`testuser@thor-x1:~/cert_no_start_line/certs$ openssl x509 -in cert.pem -text`
	`unable to load c`
	`140178126276248:`
	`bio:no start`
	`line:pem_lib.c:701:Expec`

Description:	The error message is displayed when using Transport Layer Security (TLS). The syslog-ng application uses OpenSSL for TLS and this message indicates that the certificate contains characters that OpenSSL cannot process. The error occurs when the certificate comes from Windows and you want to use it on a Linux-based computer. On Windows, the end of line (EOL) character is different (\r\n) compared to Linux (\n). To verify this, open the certificate in a text editor, for example, MCEdit. Notice the `^M` characters as shown in the image below: **Figure 45: Example of OpenSSL character processing error** 
Solution:	• On Windows, save the certificate using UTF-8, for example, using Notepad++. NOTE: Windows Notepad is not able to save the file in normal UTF-8, even if you select it. 1. In Notepad++, from the menu, select **Encoding**.

2. Change the value from **UTF-8-BOM** to **UTF-8**.
3. Save.

- On Linux, run dos2unix cert.pem. This will convert the file to a Linux-compatible style.

  Alternatively, replace the EOL characters in the file manually.

TID is already used

Error message:	`TID is already used; proto='0x202c6c0', TID='61b6456d2f02052780d0d8930cbd043857c2463fcb6014b748b1450595a6-82', client='10.140.35 Syslog connection closed;`
Description:	When a client using Advanced Log Transfer Protocol (ALTP) connects to the server for the first time, it generates a persistent ID and sends it to the server during the handshake process. This is the *TID*. If the client loses the connection to the server silently, for example, the UTP cable is pulled from the host or other network issues happen, the server is unable to detect the connection loss. If the client tries to reconnect within a short time interval, it will send the same TID. However, the server allows only one connection with the same TID. As the server "thinks" that it already has a live connection with this TID, it drops the new connection due to the duplicated TID.
Solution:	This error is eliminated automatically because the ALTP server will close the connection if there were no new messages from the client within the timeout frame. Once the timeout period of the ALTP server has passed, the client will be able to reconnect to the server (when the `time_reopen()` of the client has elapsed). If this error message appears regularly, it means that your network may be unstable, and sometimes the client loses the connection to the server in an abnormal way.

20

Best practices and examples

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 sections below:

- Do not base the separation of log messages in 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 Using name resolution in syslog-ng.

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 Using name resolution in syslog-ng.
- Enable flow-control for the TCP sources. For details, see 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 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 this 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 inaccessible 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 Resolving hostnames locally.

NOTE: Domain name resolution is important mainly in relay and server mode.

Resolving hostnames locally

Resolving hostnames locally enables you to display hostnames in the log files for frequently used hosts, without having to rely on a DNS server. The known IP address – hostname pairs are stored locally in a file. In the log messages, syslog-ng will replace the IP addresses of known hosts with their hostnames.

To configure local name resolution

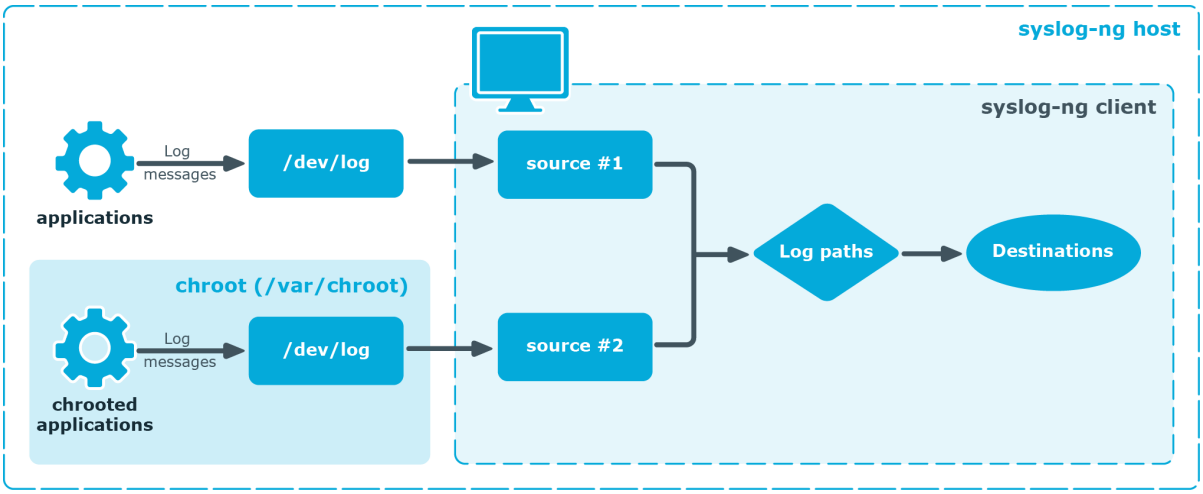
1. Add the hostnames and the respective IP addresses to the file used for local name resolution. On Linux and UNIX systems, this is the `/etc/hosts` file. Consult the documentation of your operating system for details.
2. Instruct syslog-ng to resolve hostnames locally. Set the `use-dns()` option of 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); };
```

Collecting logs from chroot

The following describes how to collect logs from a chroot using a syslog-ng client running on the host.

Figure 46: Collecting logs from chroot



To collect logs from a chroot using a syslog-ng client running on the host

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`.

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

- It is 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, are lost with each restart.

Separate incoming logs based on time, host or other information

- It is 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: 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: 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" create-dirs(yes)); };
```

Example: 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 {} \;
```

Load balancing logs between multiple destinations

These sections describe a method of load balancing logs between multiple syslog-ng Premium Edition (syslog-ng PE) destinations. The first subsection describes the round robin load balancing method based on the `R_MSEC` macro of syslog-ng PE, while the second subsection describes a configuration generator that you can use as an alternative to using the example configuration described in the first subsection.

For more information about the `R_MSEC` macro and further macros of syslog-ng PE, see Macros of syslog-ng PE.

Load balancing with a round robin load balancing method based on the R_MSEC macro of syslog-ng PE

This section describes a round robin load balancing method based on the R_MSEC macro of syslog-ng Premium Edition (syslog-ng PE) to load balance your logs between multiple syslog-ng PE destinations.

TIP: If R_MSEC is not precise enough, you can replace it with R_USEC (which uses micro-seconds instead of milliseconds).

For more information about the R_MSEC macro and further macros of syslog-ng PE, see Macros of syslog-ng PE.

Example: round robin load balancing between multiple destinations

The following example is a round-robin load balancing method, based on syslog-ng PE's R_MSEC macro.

```
destination d_lb_network {
  channel {
    channel {
      filter {
      "0" == "$(% ${R_MSEC} 2)"
      };
      destination {
        network("myhost1"
          disk-buffer(mem-buf-length(10000) disk-buf-size(2000000)));
      };
      flags(final);
    };

    channel {
    filter {
    "1" == "$(% ${R_MSEC} 2)"
    };

    destination {
      network("myhost2"
        disk-buffer(mem-buf-length(10000) disk-buf-size(2000000)));
```

```
    };
    flags(final);
    };
  };
};
```

The `filter {" <return value >" == "$(% ${R_MSEC} 2)"};` code snippets (in bold) serve as the basis of the method. This filter separates incoming log messages' timestamp values based on the `R_MSEC` macro, using a division with remainder method, and distributes the log messages equally between two destinations based on the return value (in this case, `0` or `1`).

If you need a file instead of a network destination, replace the network destination with the file in the example (and use the same analogy for any other syslog-ng PE destinations).

For an alternative method to use the round robin load balancing method based on the `R_MSEC` macro, see Configuration generator for the load balancing method based on MSEC hashing.

Configuration generator for the load balancing method based on MSEC hashing

This section describes a configuration generator for the load balancing method based on MSEC hashing to load balance your logs between multiple syslog-ng Premium Edition (syslog-ng PE) destinations.

CAUTION:

Consider that `network-load-balancer()` is not a destination, only a script that generates the example configuration described in Load balancing with a round robin load balancing method based on the R_MSEC macro of syslog-ng PE .

Also consider that the configuration generator script may change incompatibly in the future. As a result, One Identity does not officially support using this script, and recommends that you only use this script at your own risk.

As an alternative to using the example configuration described in Load balancing with a round robin load balancing method based on the R_MSEC macro of syslog-ng PE , a configuration generator script is also available in syslog-ng PE:

```
destination d_lb {network-load-balancer(targets(myhost1 myhost2 myhost3))};
```

Where destinations share the same configuration except for the destination address, balancing is based on MSEC hashing.

21

The syslog-ng manual pages

This chapter collects the manual pages of syslog-ng PE and other related applications that are usually distributed and packaged together with the syslog-ng Premium Edition application.

The dqtool tool manual page

Table of Contents

Name

dqtool — Display the contents of a disk-buffer file created with syslog-ng Premium Edition

Synopsis

`dqtool` [command] [options]

Description

NOTE: The dqtool application is distributed with the syslog-ng Premium Edition system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at the syslog-ng page.

This manual page is only an abstract, for the complete documentation of syslog-ng, see the syslog-ng Documentation page.

The **dqtool** application is a utility that can be used to display and format the messages stored in a disk-buffer file.

The cat command

`cat` [options] [file]

Use the **cat** command to display the log messages stored in the disk-buffer (also called disk-queue) file, and also information from the header of the disk queue file. The messages are printed to the standard output (stdout), so it is possible to use grep and other tools to find particular log messages, e.g., **dqtool cat /var/log/messages.qf |grep 192.168.1.1**.

The **cat** command has the following options:

--debug or **-d**

Print diagnostic and debugging messages to stderr.

--help or **-h**

Display a brief help message.

--template=<template> or **-t**

Format the messages using the specified template.

--verbose or **-v**

Print verbose messages to stderr.

--version or **-V**

Display version information.

Example:

```
        ./dqtool cat ../var/syslog-ng-00000.qf
```

The output looks like:

```
        Disk-buffer state loaded; filename='../var/syslog-ng-00000.qf', qout_
length='65', qbacklog_length='0', qoverflow_length='9205', qdisk_length='0'
Mar  3 10:52:05 tristram localprg[1234]: seq: 0000011630, runid: 1267609923,
stamp: 2010-03-03T10:52:05
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
PADDPADDPADDPADDPADDPADD
Mar  3 10:52:05 tristram localprg[1234]: seq: 0000011631, runid: 1267609923,
stamp: 2010-03-03T10:52:05
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
PADDPADDPADDPADDPADDPADD
```

Files

`/opt/syslog-ng/bin/dqtool`

See also

syslog-ng.conf(5)

syslog-ng(8)

Note

For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**

If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.

For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The logstore tool manual page

Table of Contents

Name

lgstool — Inspect and validate the binary log files (logstores) created with syslog-ng Premium Edition

Synopsis

`lgstool` [command] [options]

Description

NOTE: The lgstool application is distributed with the syslog-ng Premium Edition system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at the syslog-ng page.

This manual page is only an abstract, for the complete documentation of syslog-ng, see the syslog-ng Documentation page.

The **lgstool** application is a utility that can be used to:

- Display and format the messages stored in logstore files
- Display the record structure of logstore files
- Process log messages from orphaned journal files and write them into logstore files
- Follow (tail) messages arriving to a logstore file real-time
- Validate the digital signature and timestamp of encrypted logstore files

The cat command

`cat` [options] [file]

Use the **cat** command to display the log messages stored in the logstore file. Log messages available in the journal file of the logstore (but not yet written to the logstore file itself) are displayed as well. The messages are printed to the standard output (stdout), so it is possible to use grep and other tools to find particular log messages, e.g., **lgstool cat /var/log/messages.lgs |grep 192.168.1.1**. Note that can also follow logstore files — for details on this feature, see the section called "The tail command".

The **cat** command has the following options:

--debug or **-d**

Print diagnostic and debugging messages to stderr.

--filter<expression> or **-i**

Only print messages matching the specified syslog-ng PE filter. All possible macros, regular expressions and logical expressions can be specified in a filter.

Example 1. lgstool cat filter

```
                  lgstool cat  -t 'host: ${HOST} program: ${PROGRAM}
msg: ${MSG}\n' --filter='program("prg0000[0]")' /tmp/logstore-
serialized.lgs
```

--help or **-h**

Display a brief help message.

--key=<keyfile> or **-k**

Use the specified private key to decrypt encrypted logstore files.

--seek=<ID> or **-s**

Display only messages newer than the message specified.

--template=<template> or **-t**

Format the messages using the specified template.

--verbose or **-v**

Print verbose messages to stderr.

--version or **-V**

Display version information.

Example:

```
lgstool cat --key=mykey.pem mylogstore.lgs
```

The inspect command

`inspect` [options] [file]

Use the **inspect** command to display structure of the logstore file. The following information is displayed:

- *cipher*: The cipher algorithm used to encrypt the logstore file.
- *digest*: The digest (hash) algorithm used.
- *encrypt*: **TRUE** if the logstore file is encrypted.
- *compress*: **TRUE** if the logstore file is compressed.
- *hmac*: **TRUE** if the logstore file includes HMAC (Hash-based Message Authentication Code) information for the chunks.
- *chunk_mac*: The MAC (Message Authentication Code) of the chunk.
- *file_mac*: The MAC (Message Authentication Code) of the chunk.

For timestamped logstore files, the following information is also displayed:

- *chunk_id*: The ID of the chunk.
- *Version*: The version of the logstore file format used.
- *Policy OID*: The OID of the timestamping policy used in the timestamping request.
- *Hash Algorithm*: The digest (hash) algorithm used to create the hash of the chunk.
- *Serial number*: The serial number of the timestamp.
- *Timestamp*: The date when the Timestamping Authority timestamped the chunk.
- *Accuracy*: The accuracy of the timestamp.
- *Ordering*: Indicates the status of the ordering field in the timestamping request.
- *Nonce*: The nonce (a large random number with a high probability that it is generated by the client only once) included in the timestamping request (if any).
- *TSA*: The Distinguished Name (DN) of the Timestamping Authority.

The **inspect** command has the following options:

--debug or **-d**

Print diagnostic and debugging messages to stderr.

--help or **-h**

Display a brief help message.

--key=<keyfile> or **-k**

Use the specified private key to decrypt encrypted logstore files.

--verbose or **-v**

Print verbose messages to stderr.

--version or **-V**

Display version information.

Example:

```
lgstool inspect --key=mykey.pem mylogstore.lgs
```

A sample output looks like this:

```
XFRM_INFO @941
    cipher: aes-128-cbc
    digest: sha1
CHUNK 0@1079: [1 - 1000]:
    encrypt: TRUE
    compress: TRUE
    hmac: TRUE
    chunk_mac: e4d5d813979cf865d5ae4624f7aa98047123cd52
    file_mac: 6600600ca5befb002a73b15be8f0ac04973d5936
TIMESTAMP @36481:
    chunk_id: 0
    Status info:
    Status: Granted.
    Status description: unspecified
    Failure info: unspecified
    TST info:
    Version: 1
    Policy OID: 1.2.3.4
    Hash Algorithm: sha1
    Message data:
        0000 - 66 00 60 0c a5 be fb 00-2a 73 b1 5b e8 f0 ac 04 f.`.....*s.[....
        0010 - 97 3d 59 36                                      .=Y6
    Serial number: 0x029A
    Time stamp: Mar 19 13:48:57 2010 GMT
    Accuracy: 0x01 seconds, 0x01F4 millis, 0x64 micros
    Ordering: no
    Nonce: 0xB613F55AEFFA6DC0
    TSA: unspecified
    Extensions:
```

The recover command

`recover` [options] [file]

Warning

Do NOT use the **lgstool recover** command on logstore files that are actively used by syslog-ng PE. It might lead to data loss. Always stop syslog-ng PE first.

Use the **recover** command can process and correct broken logstore files. It can also process orphaned journal files and move their contents to the respective logstore file. Encrypted, compressed, and timestamped logstore files can be recovered as well — the private key of the logstore is not needed to recover encrypted logstore files (recovering the encrypted file does not give access to its contents). Note that the **recover** option is not available in the Windows-version of **lgstool**.

Warning

The **lgstool** application cannot fetch timestamps to the chunks (message blocks), so chunks recovered with **lgstool** are not timestamped (the internal timestamp of the syslog messages is included in the messages).

The **recover** command has the following options:

--compress-level or **-c**

Set the level of compression when processing a journal file into a compressed logstore. Default value: 3

--debug or **-d**

Print diagnostic and debugging messages to stderr.

--help or **-h**

Display a brief help message.

--verbose or **-v**

Print verbose messages to stderr.

--version or **-V**

Display version information.

Example:

```
lgstool recover mylogstore.lgs
```

The tail command

`tail` [options] [file]

Use the **tail -f** command to follow the contents of a logstore file like the traditional **tail** command does on Linux/UNIX systems. The messages are printed to the standard output (stdout). Contents of the journal file related to the logstore file are displayed as well.

The **tail** command has the following options.

--debug or **-d**

Print diagnostic and debugging messages to stderr.

--help or **-h**

Display a brief help message.

--filter=<expression> or **-i**

Only print messages matching the specified syslog-ng PE filter. All possible macros, regular expressions and logical expressions can be specified in a filter.

Example 2. lgstool tail filter

```
lgstool tail  -t 'host: ${HOST} program: ${PROGRAM}
msg: ${MSG}\n' --filter='program("prg0000[0]")' /tmp/logstore-
serialized.lgs
```

--follow or **-f**

Follow mode: display messages as they arrive into the logstore.

--key=<keyfile> or **-k**

Use the specified private key to decrypt encrypted logstore files.

--lines=<N> or **-n**

Display the last N lines of the logstore file instead of the last 10. Alternatively, use **+N** to display lines starting with the Nth.

--sleep_interval=<seconds> or **-s**

Number of seconds to wait before displaying new messages in follow mode.

--template=<template> or **-t**

Format the messages using the specified template.

--verbose or **-v**

Print verbose messages to stderr.

--version or **-V**

Display version information.

Example:

```
lgstool tail -f -n=20 --key=mykey.pem mylogstore.lgs
```

The validate command

`validate` [options] [file]

Use the **validate** command to validate the signatures and timestamps of a logstore file. The **validate** command has the following options:

--ca-dir=<directory> or **-C**

The directory that stores the certificates of the trusted Certificate Authorities. Use this option if the timestamps of your logstore files were signed with certificates belonging to different Certificate Authorities.

--ca-dir-layout=<md5|sha1>

The type of the hash used for the CA certificates. The default value (**md5**) is expected to change to **sha1** in subsequent releases of syslog-ng PE.

--ca-file=<file> or **-P**

A file that stores the certificate of the trusted Certificate Authority. Use this option if the timestamps of your logstore files were signed with a single certificate, or if every such certificate belongs to the same Certificate Authority.

--crl-dir=<directory> or **-R**

The directory that stores the Certificate Revocation Lists of the trusted Certificate Authorities.

--debug or **-d**

Print diagnostic and debugging messages to stderr.

--help or **-h**

Display a brief help message.

--key=<keyfile> or **-k**

Use the specified private key to decrypt encrypted logstore files.

--require-ts or **-T**

Consider the logstore file invalid unless the entire file is protected by a valid timestamp.

--seed or **-S**

Use the `~/.rnd` file or the file specified in the *$RANDFILE* environmental variable as seed. This is needed only on platforms that do not have a `/dev/random` device (for example, Solaris) and the entropy gathering daemon **egd** application is not installed on the system.

--ts-name=<name> or **-D**

Consider the logstore file invalid unless the timestamps are signed by the specified Timestamping Authority. Specify the Distinguished Name (DN) of the Timestamping Authority.

--verbose or **-v**

Print verbose messages to stderr.

--version or **-V**

Display version information.

By default, the **lgstool validate** command checks only the checksum of the file. Use the *--require-ts* option to validate the timestamps as well. The digital signature of the timestamps is checked only if the *--ca-dir* or the *--ca-file* parameter is set.

Example:

```
        lgstool validate --key=mykey.pem --ca-file=mycacert.pem --ts-name=MYTSA
mylogstore.lgs
```

The reindex command

`reindex` [options] [file]

The **reindex** command is an experimental, currently unsupported tool. Do not attempt to use it unless your syslog-ng PE support team explicitly instructs you to do so.

Files

`/opt/syslog-ng/bin/lgstool`

See also

syslog-ng.conf(5)

syslog-ng(8)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**
>
> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The loggen manual page

Table of Contents

Name

loggen — Generate syslog messages at a specified rate

Synopsis

`loggen` [options]
`target` [port]

Description

NOTE: The loggen application is distributed with the syslog-ng system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at the syslog-ng page.

This manual page is only an abstract, for the complete documentation of syslog-ng, see the syslog-ng Documentation page.

The **loggen** application is tool to test and stress-test your syslog server and the connection to the server. It can send syslog messages to the server at a specified rate, using a number of connection types and protocols, including TCP, UDP, and unix domain sockets. The messages can be generated automatically (repeating the **PADD**string over and over), or read from a file or the standard input. The following is a sample generated message:

```
        <38>2017-04-05T12:16:46 localhost prg00000[1234]: seq: 0000000000,
thread: 0000, runid: 1491387406, stamp: 2017-04-05T12:16:46
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
```

When **loggen** finishes sending the messages, it displays the following statistics:

- *average rate*: Average rate the messages were sent in messages/second.
- *count*: The total number of messages sent.
- *time*: The time required to send the messages in seconds.
- *average message size*: The average size of the sent messages in bytes.
- *bandwidth*: The average bandwidth used for sending the messages in kilobytes/second.

Options

--active-connections <number-of-connections>

Number of connections **loggen** will use to send messages to the destination. This option is usable only when using TCP or TLS connections to the destination. Default value: 1

The **loggen** utility waits until every connection is established before starting to send messages. See also the `--idle-connections` option.

--csv or **-C**

Send the statistics of the sent messages to stdout as CSV. This can be used for plotting the message rate.

--dgram or **-D**

Use datagram socket (UDP or unix-dgram) to send the messages to the target. Requires the *--inet* option as well.

--dont-parse or **-d**

Do not parse the lines read from the input files, send them as received.

--help or **-h**

Display a brief help message.

--idle-connections <number-of-connections>

Number of idle connections **loggen** will establish to the destination. Note that **loggen** will not send any messages on idle connections, but the connection is kept open using keep-alive messages. This option is usable only when using TCP or TLS connections to the destination. See also the *--active-connections* option. Default value: 0

--inet or **-i**

Use the TCP (by default) or UDP (when used together with the *--dgram* option) protocol to send the messages to the target.

--interval <seconds> or **-I <seconds>**

The number of seconds **loggen** will run. Default value: 10

> **Note**
>
> Note that when the *--interval* and *--number* are used together, **loggen** will send messages until the period set in *--interval* expires or the amount of messages set in *--number* is reached, whichever happens first.

--ipv6 or **-6**

Specify the destination using its IPv6 address. Note that the destination must have a real IPv6 address.

--loop-reading or **-l**

Read the file specified in *--read-file* option in loop: loggen will start reading from the beginning of the file when it reaches the end of the file.

--number <number-of-messages> or **-n <number-of-messages>**

Number of messages to generate.

> **Note**
>
> Note that when the *--interval* and *--number* are used together, **loggen** will send messages until the period set in *--interval* expires or the amount of messages set in *--number* is reached, whichever happens first.

--no-framing or **-F**

Do not use the framing of the IETF-syslog protocol style, even if the *--syslog-proto* option is set.

--quiet or **-Q**

Output statistics only when the execution of **loggen** is finished. If not set, the statistics are displayed every second.

--permanent or **-T**

Keep sending logs indefinitely, without time limit.

--rate <message/second> or **-r <message/second>**

The number of messages generated per second for every active connection. Default value: 1000

If you want to change the message rate while loggen is running, send SIGUSR1 to double the message rate, or SIGUSR2 to halve it:
kill -USR1 <loggen-pid> kill -USR2 <loggen-pid>

--read-file <filename> or **-R <filename>**

Read the messages from a file and send them to the target. See also the *--skip-tokens* option.

Specify – as the input file to read messages from the standard input (stdio). Note that when reading messages from the standard input, **loggen** can only use a single thread. The **-R** – parameters must be placed at end of command, like: **loggen 127.0.0.1 1061 --read-file -**

--sdata <data-to-send> or **-p <data-to-send>**

Send the argument of the *--sdata* option as the SDATA part of IETF-syslog (RFC5424 formatted) messages. Use it together with the *--syslog-proto* option. For example: **--sdata "[test name=\"value\"]**

--size <message-size> or **-s <message-size>**

The size of a syslog message in bytes. Default value: 256. Minimum value: 127 bytes, maximum value: 8192 bytes.

--skip-tokens <number>

Skip the specified number of space-separated tokens (words) at the beginning of every line. For example, if the messages in the file look like **foo bar message**, **--skip-tokens 2** skips the **foo bar** part of the line, and sends only the **message** part. Works only when used together with the *--read-file* parameter. Default value: 0

--stream or **-S**

Use a stream socket (TCP or unix-stream) to send the messages to the target.

--syslog-proto or **-P**

Use the new IETF-syslog message format as specified in RFC5424. By default, loggen uses the legacy BSD-syslog message format (as described in RFC3164). See also the *--no-framing* option.

--unix </path/to/socket> or **-x </path/to/socket>**

Use a UNIX domain socket to send the messages to the target.

--use-ssl or **-U**

Use an SSL-encrypted channel to send the messages to the target. Note that it is not possible to check the certificate of the target, or to perform mutual authentication.

--version or **-V**

Display version number of syslog-ng.

Examples

The following command generates 100 messages per second for ten minutes, and sends them to port 2010 of the localhost via TCP. Each message is 300 bytes long.

```
loggen --size 300 --rate 100 --interval 600 127.0.0.1 2010
```

The following command is similar to the one above, but uses the UDP protocol.

```
loggen --inet --dgram --size 300 --rate 100 --interval 600
127.0.0.1 2010
```

Send a single message on TCP6 to the `::1` IPv6 address, port `1061`:

```
loggen --ipv6 --number 1 ::1 1061
```

Send a single message on UDP6 to the `::1` IPv6 address, port `1061`:

```
loggen --ipv6 --dgram --number 1 ::1 1061
```

Send a single message using a unix domain-socket:

```
loggen --unix --stream --number 1 </path/to/socket>
```

Read messages from the standard input (stdio) and send them to the localhost:

```
loggen 127.0.0.1 1061 --read-file -
```

Files

`/opt/syslog-ng/bin/loggen`

See also

syslog-ng.conf(5)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**

> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The pdbtool manual page

Table of Contents

Name

pdbtool — An application to test and convert syslog-ng pattern database rules

Synopsis

`pdbtool` [command] [options]

Description

This manual page is only an abstract, for the complete documentation of syslog-ng and pdbtool, see the syslog-ng Documentation page.

The syslog-ng application can match the contents of the log messages to a database of predefined message patterns (also called patterndb). By comparing the messages to the known patterns, syslog-ng is able to identify the exact type of the messages, tag the messages, and sort them into message classes. The message classes can be used to classify the type of the event described in the log message. The functionality of the pattern database is similar to that of the logcheck project, but the syslog-ng approach is faster, scales better, and is much easier to maintain compared to the regular expressions of logcheck.

The **pdbtool** application is a utility that can be used to:

- test messages, or specific rules
- convert an older pattern database to the latest database format
- merge pattern databases into a single file
- automatically create pattern databases from a large amount of log messages
- dump the RADIX tree built from the pattern database (or a part of it) to explore how the pattern matching works.

The dictionary command

`dictionary` [options]

Lists every name-value pair that can be set by the rules of the pattern database.

--dump-tags or **-T**

List the tags instead of the names of the name-value pairs.

--pdb <path-to-file> or **-p <path-to-file>**

Name of the pattern database file to use.

--program <programname> or **-P <programname>**

List only the name-value pairs that can be set for the messages of the specified *$PROGRAM* application.

The dump command

`dump` [options]

Display the RADIX tree built from the patterns. This shows how are the patterns represented in syslog-ng and it might also help to track down pattern-matching problems. The dump utility can dump the tree used for matching the PROGRAM or the MSG parts.

--debug or **-d**

Enable debug/diagnostic messages on stderr.

--pdb or **-p**

Name of the pattern database file to use.

--program or **-P**

Displays the RADIX tree built from the patterns belonging to the *${PROGRAM}* application.

--program-tree or **-T**

Display the *${PROGRAM}* tree.

--verbose or **-v**

Enable verbose messages on stderr.

Example and sample output:

```
        pdbtool dump -p patterndb.xml  -P 'sshd'
```

```
'p'
   'assword for'
     @QSTRING:@
       'from'
        @QSTRING:@
          'port '
            @NUMBER:@ rule_id='fc49054e-75fd-11dd-9bba-001e6806451b'
              ' ssh' rule_id='fc55cf86-75fd-11dd-9bba-001e6806451b'
                '2' rule_id='fc4b7982-75fd-11dd-9bba-001e6806451b'
    'ublickey for'
      @QSTRING:@
        'from'
         @QSTRING:@
           'port '
             @NUMBER:@ rule_id='fc4d377c-75fd-11dd-9bba-001e6806451b'
               ' ssh' rule_id='fc5441ac-75fd-11dd-9bba-001e6806451b'
                 '2' rule_id='fc44a9fe-75fd-11dd-9bba-001e6806451b'
```

The match command

`match` [options]

Use the **match** command to test the rules in a pattern database. The command tries to match the specified message against the patterns of the database, evaluates the parsers of the pattern, and also displays which part of the message was parsed successfully. The command returns with a `0` (success) or `1` (no match) return code and displays the following information:

- the class assigned to the message (that is, system, violation, and so on),
- the ID of the rule that matched the message, and
- the values of the parsers (if there were parsers in the matching pattern).

The **match** command has the following options:

--color-out or **-c**

Color the terminal output to highlight the part of the message that was successfully parsed.

--debug or **-d**

Enable debug/diagnostic messages on stderr.

--debug-csv or **-C**

Print the debugging information returned by the *`--debug-pattern`* option as comma-separated values.

--debug-pattern or **-D**

Print debugging information about the pattern matching. See also the `--debug-csv` option.

--file=<filename-with-path> or **-f**

Process the messages of the specified log file with the pattern database. This option allows to classify messages offline, and to apply the pattern database to already existing logfiles. To read the messages from the standard input (stdin), specify a hyphen (-) character instead of a filename.

--filter=<filter-expression> or **-F**

Print only messages matching the specified syslog-ng filter expression.

--message or **-M**

The text of the log message to match (only the `${MESSAGE}` part without the syslog headers).

--pdb or **-p**

Name of the pattern database file to use.

--program or **-P**

Name of the program to use, as contained in the `${PROGRAM}` part of the syslog message.

--template=<template-expression> or **-T**

A syslog-ng template expression that is used to format the output messages.

--verbose or **-v**

Enable verbose messages on stderr.

Example: The following command checks if the `patterndb.xml` file recognizes the **`Accepted publickey for myuser from 127.0.0.1 port 59357 ssh6`** message:

```
        pdbtool match -p patterndb.xml -P sshd -M "Accepted publickey for myuser
from 127.0.0.1 port 59357 ssh6"
```

The following example applies the `sshd.pdb` pattern database file to the log messages stored in the `/var/log/messages` file, and displays only the messages that received a **`useracct`** tag.

```
pdbtool match -p sshd.pdb \
  -file /var/log/messages \
  -filter 'tags("usracct");'
```

The merge command

`merge` [options]

Use the **merge** command to combine separate pattern database files into a single file (pattern databases are usually stored in separate files per applications to simplify maintenance). If a file uses an older database format, it is automatically updated to the

latest format (V3). See the **The syslog-ng Administrator Guide** for details on the different pattern database versions.

--debug or **-d**

Enable debug/diagnostic messages on stderr.

--directory or **-D**

The directory that contains the pattern database XML files to be merged.

--glob or **-G**

Specify filenames to be merged using a glob pattern, for example, using wildcards. For details on glob patterns, see **man glob**. This pattern is applied only to the filenames, and not on directory names.

--pdb or **-p**

Name of the output pattern database file.

--recursive or **-r**

Merge files from subdirectories as well.

--verbose or **-v**

Enable verbose messages on stderr.

Example:

```
        pdbtool merge --recursive --directory /home/me/mypatterns/  --pdb
/var/lib/syslog-ng/patterndb.xml
```

Currently it is not possible to convert a file without merging, so if you only want to convert an older pattern database file to the latest format, you have to copy it into an empty directory.

The patternize command

`patternize` [options]

Automatically create a pattern database from a log file containing a large number of log messages. The resulting pattern database is printed to the standard output (stdout). The **pdbtool patternize** command uses a data clustering technique to find similar log messages and replacing the differing parts with **`@ESTRING:: @`** parsers. For details on pattern databases and message parsers, see the **The syslog-ng Administrator Guide**. The **patternize** command is available only in syslog-ng PE version 3.2 and later.

--debug or **-d**

Enable debug/diagnostic messages on stderr.

--file=<path> or **-f**

The logfile containing the log messages to create patterns from. To receive the log messages from the standard input (stdin), use –.

--iterate-outliers or **-o**

Recursively iterate on the log lines to cover as many log messages with patterns as possible.

--named-parsers or **-n**

The number of example log messages to include in the pattern database for every pattern. Default value: `1`

--no-parse or **-p**

Do not parse the input file, treat every line as the message part of a log message.

--samples=<number-of-samples>

Include a generated name in the parsers, for example, `.dict.string1`, `.dict.string2`, and so on.

--support=<number> or **-S**

A pattern is added to the output pattern database if at least the specified percentage of log messages from the input logfile match the pattern. For example, if the input logfile contains 1000 log messages and the `--support=3.0` option is used, a pattern is created only if the pattern matches at least 3 percent of the log messages (that is, 30 log messages). If patternize does not create enough patterns, try to decrease the support value.

Default value: `4.0`

--verbose or **-v**

Enable verbose messages on stderr.

Example:

```
pdbtool patternize --support=2.5 --file=/var/log/messages
```

The test command

`test` [options]

Use the **test** command to validate a pattern database XML file. Note that you must have the **xmllint** application installed. The **test** command is available only in syslog-ng PE version 3.2 and later.

--color-out or **-c**

Enable coloring in terminal output.

--debug or **-d**

Enable debug/diagnostic messages on stderr.

--debug or **-D**

Print debugging information on non-matching patterns.

--rule-id or **-r**

Test only the patterndb rule (specified by its rule id) against its example.

--validate

Validate a pattern database XML file.

--verbose or **-v**

Enable verbose messages on stderr.

Example:

```
pdbtool test --validate /home/me/mypatterndb.pdb
```

Files

`/opt/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng.conf`

See also

The syslog-ng Administrator Guide

syslog-ng.conf(5)

syslog-ng(8)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**
>
> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The persist-tool tool manual page

Name

persist-tool — Display the content of the persist file

Synopsis

`persist-tool` [command] [options]

Description

NOTE: The persist-tool application is distributed with the system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at https://syslog-ng.com..

This manual page is only an abstract, for the complete documentation of syslog-ng, see https://syslog-ng.com..

The **persist-tool** application is a utility that can be used to dump the content of the persist file, and manipulate its content.

Warning

Persist-tool is a special tool for syslog-ng experts. Do use the tool unless you know exactly what you are doing. Misconfiguring it will result in irrecoverable damage to the persist file, without any warning.

Note

Limitations:

- The persist-state functions can be used only with syslog-ng PE 5 LTS style persist file (SLP4). Older persist files are not supported.
- Wildcard characters are not supported in file/directory names.

The dump command

`dump` [options] [persist_file]

Use the **dump** command to print the current content of the persist file in JSON format to the console.

The **dump** command has the following options:

--help or **-?**

Display a brief help message.

Example:

```
persist-tool dump /opt/syslog-ng/var/syslog-ng.persist
```

The output looks like:

```
run_id = { "value": "00 00 00 00 0C 00 00 00 " }
host_id = { "value": "00 00 00 00 5F 49 2F 01 " }
```

The add command

`add` [options] [input_file]

Use the **add** command to add or modify a specified state-entry in the persist file. The state-entry should be in the same format as the **dump** command displays it. If the given state-entry already exists, it will be updated. Otherwise, a new value will be added. If the given persist state is invalid, it will be skipped.

To use the **add** command: use **persist-tool dump** to print the content of the current persist file, and redirect it to a file. Edit the content of this file. Use **persist-tool add** with this file to modify the persist.

The **add** command has the following options:

--help or **-?**

Display a brief help message.

--output-dir=<directory> or **-o**

Required parameter. The directory where the persist file is located at. The name of the persist file stored in this directory must be `syslog-ng.persist`.

--persist-name=<filename> or **-p**

Optional parameter. The name of the persist file to generate. Default value: `syslog-ng.persist`.

Example:

```
/opt/syslog-ng/bin/persist-tool add dump_persist -o .
```

The valid output looks like:

```
log_reader_curpos(Application)      OK
affile_sd_curpos(/var/aaa.txt)        OK
```

The invalid output looks like:

```
log_reader_curpos(Application)      OK
wrong
        FAILED (error: Invalid entry syntax)
affile_sd_curpos(/var/aaa.txt)        OK
```

Files

`/opt/syslog-ng/bin/persist-tool`

See also

syslog-ng.conf(5)

syslog-ng(8)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**
>
> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The syslog-debun manual page

Table of Contents

Name

syslog-debun — syslog-ng DEBUg buNdle generator

Synopsis

`syslog-debun` [options]

Description

NOTE: The **syslog-debun** application is distributed with the syslog-ng PE system logging application, and is usually part of the syslog-ng PE package. The latest version of the syslog-ng PE application is available at the syslog-ng page.

This manual page is only an abstract, for the complete documentation of syslog-ng, see the syslog-ng Documentation page.

The **syslog-debun** tool collects and saves information about your syslog-ng PE installation, making troubleshooting easier, especially if you ask help about your syslog-ng PE related problem.

General Options

-h

Display the help page.

-l

Do not collect privacy-sensitive data, for example, process tree, fstab, and so on. If you use with **-d**, then the following parameters will be used for debug mode: **-Fev**

-R <directory>

The directory where syslog-ng PE is installed instead of `/opt/syslog-ng`.

-W <directory>

Set the working directory, where the debug bundle will be saved. Default value: `/tmp`. The name of the created file is **syslog.debun.${host}.${date}.${3-random-characters-or-pid}.tgz**

Debug mode options

-d

Start syslog-ng PE in debug mode, using the **-Fedv --enable-core** options.

Warning! Using this option under high message load may increase disk I/O during the debug, and the resulting debug bundle can be huge. To exit debug mode, press Enter.

-D <options>

Start syslog-ng PE in debug mode, using the specified command-line options. To exit debug mode, press Enter. For details on the available options, see ???.

-t <seconds>

Run syslog-ng PE in noninteractive debug mode for <seconds>, and automatically exit debug mode after the specified number of seconds.

-w <seconds>

Wait <seconds> seconds before starting debug mode.

System call tracing

-s

Enable syscall tracing (**strace -f** or **truss -f**). Note that using `-s` itself does not enable debug mode, only traces the system calls of an already running syslog-ng PE process. To trace system calls in debug mode, use both the `-s` and `-d` options.

Packet capture options

Capturing packets requires a packet capture tool on the host. The **syslog-debun** tool attempts to use **tcpdump** on most platforms, except for Solaris, where it uses **snoop**.

-i <interface>

Capture packets only on the specified interface, for example, `eth0`.

-p

Capture incoming packets using the following filter: `port 514 or port 601 or port 53`

-P <options>

Capture incoming packets using the specified filter.

-t <seconds>

Run syslog-ng PE in noninteractive debug mode for <seconds>, and automatically exit debug mode after the specified number of seconds.

Examples

```
syslog-debun
```

Create a simple debug bundle, collecting information about your environment, for example, list packages containing the word: syslog, ldd of your syslog-binary, and so on.

```
syslog-debun -l
```

Similar to **syslog-debun**, but without privacy-sensitive information. For example, the following is NOT collected: fstab, df output, mount info, ip / network interface configuration, DNS resolv info, and process tree.

```
syslog-debun -d
```

Similar to **syslog-debun**, but it also stops syslog-ng, then restarts it in debug mode (`-Fedv --enable-core`). To stop debug mode, press Enter. The output of the debug mode collected into a separate file, and also added to the debug bundle.

```
syslog-debun -s
```

Trace the system calls (using **strace** or **truss**) of an already running syslog-ng PE process.

```
syslog-debun -d -s
```

Restart syslog-ng PE in debug mode, and also trace the system calls (using **strace** or **truss**) of the syslog-ng PE process.

```
syslog-debun -p
```

Run packet capture (pcap) with the filter: **`port 514 or port 601 or port 53`** Also waits for pressing Enter, like debug mode.

```
syslog-debun -p -t 10
```

Noninteractive debug mode: Similar to **syslog-debun -p**, but automatically exit after 10 seconds.

```
        syslog-debun -P "host 1.2.3.4"  -D "-Fev --enable-core"
```

Change the packet-capturing filter from the default to **`host 1.2.3.4`**. Also change debugging parameters from the default to **`-Fev --enable-core`**. Since a timeout (**`-t`**) is not given, waits for pressing Enter.

```
syslog-debun -p -d -w 5 -t 10
```

Collect pcap and debug mode output following this scenario:

- Start packet capture with default parameters (**`-p`**)
- Wait 5 seconds (**`-w 5`**)
- Stop syslog-ng
- Start syslog-ng in debug mode with default parameters (**`-d`**)
- Wait 10 seconds (**`-t 10`**)
- Stop syslog-ng debuging
- Start syslog-ng
- Stop packet capturing

Files

`/opt/syslog-ng/bin/loggen`

See also

syslog-ng.conf(5)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**

> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The syslog-ng control tool manual page

Table of Contents

Name

syslog-ng-ctl — Display message statistics and enable verbose, debug and trace modes in syslog-ng Premium Edition

Synopsis

`syslog-ng-ctl` [command] [options]

Description

NOTE: The syslog-ng-ctl application is distributed with the syslog-ng Premium Edition system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at syslog-ng page.

This manual page is only an abstract, for the complete documentation of syslog-ng, see the syslog-ng Documentation page.

The **syslog-ng-ctl** application is a utility that can be used to:

- enable/disable various syslog-ng messages for troubleshooting
- display statistics about the processed messages

- handling password-protected private keys
- display the currently running configuration of syslog-ng PE
- reload the configuration of syslog-ng PE.
- stop syslog-ng PE.

Enabling troubleshooting messages

`command` [options]

Use the **syslog-ng-ctl <command> --set=on** command to display verbose, trace, or debug messages. If you are trying to solve configuration problems, the verbose (and occasionally trace) messages are usually sufficient. Debug messages are needed mostly for finding software errors. After solving the problem, do not forget to turn these messages off using the **syslog-ng-ctl <command> --set=off**. Note that enabling debug messages does not enable verbose and trace messages.

Use **syslog-ng-ctl <command>** without any parameters to display whether the particular type of messages are enabled or not.

If you need to use a non-standard control socket to access syslog-ng, use the **syslog-ng-ctl <command> --set=on --control=<socket>** command to specify the socket to use.

verbose

Print verbose messages. If syslog-ng was started with the *--stderr* or **-e** option, the messages will be sent to stderr. If not specified, syslog-ng will log such messages to its internal source.

trace

Print trace messages of how messages are processed. If syslog-ng was started with the *--stderr* or **-e** option, the messages will be sent to stderr. If not specified, syslog-ng will log such messages to its internal source.

debug

Print debug messages. If syslog-ng was started with the *--stderr* or **-e** option, the messages will be sent to stderr. If not specified, syslog-ng will log such messages to its internal source.

Example:

```
syslog-ng-ctl verbose --set=on
```

syslog-ng-ctl query

The syslog-ng PE application stores various data, metrics, and statistics in a hash table. Every property has a name and a value. For example:

```
[syslog-ng]
|
|_[destinations]-[network]-[tcp]->[stats]->{received=12;dropped=2}
|
|_[sources]-[sql]-[stats]->{received=501;dropped=0}
```

You can query the nodes of this tree, and also use filters to select the information you need. A query is actually a path in the tree. You can also use the `?` and `*` wildcards. For example:

- Select every property: `*`
- Select all **`dropped`** value from every **`stats`** node: **`*.stats.dropped`**

The nodes and properties available in the tree depend on your syslog-ng PE configuration (that is, the sources, destinations, and other objects you have configured), and also on your *`stats-level()`* settings.

The list command

```
syslog-ng-ctl query list
```

Use the **syslog-ng-ctl query list** command to display the list of metrics that syslog-ng PE collects about the processed messages. For details about the displayed metrics, see The syslog-ng Administrator Guide???.

An example output:

```
center.received.stats.processed
center.queued.stats.processed
destination.java.d_elastic#0.java_dst(ElasticSearch,elasticsearch-syslog-ng-
test,t7cde889529c034aea9ec_micek).stats.dropped
destination.java.d_elastic#0.java_dst(ElasticSearch,elasticsearch-syslog-ng-
test,t7cde889529c034aea9ec_micek).stats.processed
destination.java.d_elastic#0.java_dst(ElasticSearch,elasticsearch-syslog-ng-
test,t7cde889529c034aea9ec_micek).stats.queued
destination.d_elastic.stats.processed
source.s_tcp.stats.processed
source.severity.7.stats.processed
source.severity.0.stats.processed
source.severity.1.stats.processed
source.severity.2.stats.processed
source.severity.3.stats.processed
source.severity.4.stats.processed
source.severity.5.stats.processed
source.severity.6.stats.processed
source.facility.7.stats.processed
source.facility.16.stats.processed
source.facility.8.stats.processed
source.facility.17.stats.processed
source.facility.9.stats.processed
source.facility.18.stats.processed
source.facility.19.stats.processed
```

```
source.facility.20.stats.processed
source.facility.0.stats.processed
source.facility.21.stats.processed
source.facility.1.stats.processed
source.facility.10.stats.processed
source.facility.22.stats.processed
source.facility.2.stats.processed
source.facility.11.stats.processed
source.facility.23.stats.processed
source.facility.3.stats.processed
source.facility.12.stats.processed
source.facility.4.stats.processed
source.facility.13.stats.processed
source.facility.5.stats.processed
source.facility.14.stats.processed
source.facility.6.stats.processed
source.facility.15.stats.processed
source.facility.other.stats.processed
global.payload_reallocs.stats.processed
global.msg_clones.stats.processed
global.sdata_updates.stats.processed
tag..source.s_tcp.stats.processed
```

The **syslog-ng-ctl query list** command has the following options:

--reset

Use **--reset** to set the selected counters to 0 after executing the query.

Displaying metrics and statistics

`syslog-ng-ctl query get` [options]

The **syslog-ng-ctl query get <query>** command lists the nodes that match the query, and their values.

For example, the **`"destination*"`** query lists the configured destinations, and the metrics related to each destination. An example output:

```
         destination.java.d_elastic#0.java_dst(ElasticSearch,elasticsearch-
syslog-ng-test,t7cde889529c034aea9ec_micek).stats.dropped=0
destination.java.d_elastic#0.java_dst(ElasticSearch,elasticsearch-syslog-ng-
test,t7cde889529c034aea9ec_micek).stats.processed=0
destination.java.d_elastic#0.java_dst(ElasticSearch,elasticsearch-syslog-ng-
test,t7cde889529c034aea9ec_micek).stats.queued=0
destination.d_elastic.stats.processed=0
```

The **syslog-ng-ctl query get** command has the following options:

--sum

Add up the result of each matching node and return only a single number.

For example, the **`syslog-ng-ctl query get --sum "destination*.dropped"`** command displays the number of messages dropped by the syslog-ng PE instance.

--reset

Use **--reset** to set the selected counters to 0 after executing the query.

The stats command

`stats` [options]

Use the **stats** command to display statistics about the processed messages. For details about the displayed statistics, see The syslog-ng Administrator Guide???. The **stats** command has the following options:

--control=<socket> or **-c**

Specify the socket to use to access syslog-ng. Only needed when using a non-standard socket.

--reset=<socket> or **-r**

Reset all statistics to zero, except for the **`queued`** counters. (The **`queued`** counters show the number of messages in the message queue of the destination driver, waiting to be sent to the destination.)

Example:

```
syslog-ng-ctl stats
```

An example output:

```
        src.internal;s_all#0;;a;processed;6445
src.internal;s_all#0;;a;stamp;1268989330
destination;df_auth;;a;processed;404
destination;df_news_dot_notice;;a;processed;0
destination;df_news_dot_err;;a;processed;0
destination;d_ssb;;a;processed;7128
destination;df_uucp;;a;processed;0
source;s_all;;a;processed;7128
destination;df_mail;;a;processed;0
destination;df_user;;a;processed;1
destination;df_daemon;;a;processed;1
destination;df_debug;;a;processed;15
destination;df_messages;;a;processed;54
destination;dp_xconsole;;a;processed;671
dst.tcp;d_network#0;10.50.0.111:514;a;dropped;5080
dst.tcp;d_network#0;10.50.0.111:514;a;processed;7128
dst.tcp;d_network#0;10.50.0.111:514;a;queued;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
```

Displaying license-related information

`syslog-ng-ctl show-license-info` [options]

The syslog-ng PE application uses a license in server mode to determine the maximum number of hosts that are allowed to connect. Use the **syslog-ng-ctl show-license-info** command to display license-related information the number of hosts currently logging to your server. This helps you to plan your capacity, to check your license usage, and to detect client misconfiguration that can result in a license miscount anomaly. Note that in client or relay mode, syslog-ng PE does not require a license.

The **syslog-ng-ctl show-license-info** command displays the following information. In case of an unlimited license, or in client or relay mode, only the license type is displayed:

- License Type: none, limited, unlimited
- Host Limit: the maximum number of hosts that are allowed to connect.
- Currently Used Slots: the number of currently used host slots
- Usage: the percent of used host slots
- Licensed Clients: the list of hostnames that are stored in the license module

The **syslog-ng-ctl show-license-info** command has the following options:

--json or **-J**

Print license-related information in JSON format.

Example:

```
syslog-ng-ctl show-license-info
```

An example output:

```
License-Type: limited
Host-Limit: 10
Currently-Used-Slots: 7
Usage: 70%
Licensed-Clients:
    192.168.0.1
    192.168.0.2
    192.168.0.3
    192.168.1.4
    192.168.1.5
```

Example:

```
        syslog-ng-ctl show-license-info --json
```

An example output:

```
{
       "license_type":  "limited",
       "host_limit":  10,
       "currently_used_slots":  7,
       "usage":  "70%",
       "licensed_clients":  [
             "xy.testdomain",
             "testhost",
             "192.168.0.3",
             "test_host",
             "192.168.1.5"
       ]
 }
```

Example:

```
syslog-ng-ctl show-license-info
```

in case of an unlimited license

An example output:

```
$ syslog-ng-ctl show-license-info
License-Type: unlimited
```

Example:

```
syslog-ng-ctl show-license-info
```

if syslog-ng PE is in client or relay mode

An example output:

```
$ syslog-ng-ctl show-license-info
License-Type: none
```

Handling password-protected private keys

`syslog-ng-ctl credentials` [options]

The **syslog-ng-ctl credentials status** command allows you to query the status of the private keys that syslog-ng PE uses in the *`network()`* and *`syslog()`* drivers. You can also provide the passphrase for password-protected private keys using the **syslog-ng-ctl**

credentials add command. For details on using password-protected keys, see The syslog-ng Administrator Guide???.

Displaying the status of private keys

`syslog-ng-ctl credentials status` [options]

The **syslog-ng-ctl credentials status** command allows you to query the status of the private keys that syslog-ng PE uses in the *network()* and *syslog()* drivers. The command returns the list of private keys used, and their status. For example:

```
syslog-ng-ctl credentials status
Secret store status:
/home/user/ssl_test/client-1/client-encrypted.key SUCCESS
```

If the status of a key is PENDING, you must provide the passphrase for the key, otherwise syslog-ng PE cannot use it. The sources and destinations that use these keys will not work until you provide the passwords. Other parts of the syslog-ng PE configuration will be unaffected. You must provide the passphrase of the password-protected keys every time syslog-ng PE is restarted.

The following log message also notifies you of PENDING passphrases:

```
        Waiting for password; keyfile='private.key'
```

--control=<socket> or **-c**

Specify the socket to use to access syslog-ng. Only needed when using a non-standard socket.

Opening password-protected private keys

`syslog-ng-ctl credentials add` [options]

You can add the passphrase to a password-protected private key file using the following command. syslog-ng PE will display a prompt for you to enter the passphrase. We recommend that you use this method.

```
        syslog-ng-ctl credentials add --id=<path-to-the-key>
```

Alternatively, you can include the passphrase in the *--secret* parameter:

```
        syslog-ng-ctl credentials add --id=<path-to-the-key> --
secret=<passphrase-of-the-key>
```

Or you can pipe the passphrase to the syslog-ng-ctl command, for example:

```
        echo "<passphrase-of-the-key>" | syslog-ng-ctl credentials add --
id=<path-to-the-key>
```

--control=<socket> or **-c**

Specify the socket to use to access syslog-ng. Only needed when using a non-standard socket.

--id=<path-to-the-key> or **-i**

The path to the password-protected private key file. This is the same path that you use in the *key-file()* option of the syslog-ng PE configuration file.

--secret=<passphrase-of-the-key> or **-s**

The password or passphrase of the private key.

Displaying the configuration

`syslog-ng-ctl config` [options]

Use the **syslog-ng-ctl config** command to display the configuration that syslog-ng PE is currently running. Note by default, only the content of the main configuration file are displayed, included files are not resolved. To resolve included files and display the entire configuration, use the **syslog-ng-ctl config --preprocessed** command.

Reloading the configuration

`syslog-ng-ctl reload` [options]

Use the **syslog-ng-ctl reload** command to reload the configuration file of syslog-ng PE without having to restart the syslog-ng PE application. The **syslog-ng-ctl reload** works like a SIGHUP (-1). On Microsoft Windows, this is the only way to reload the configuration of syslog-ng PE.

The syslog-ng-ctl reload command returns 0 if the operation was successful, 1 otherwise.

Stopping syslog-ng PE

`syslog-ng-ctl stop` [options]

Use the **syslog-ng-ctl stop** command to stop the syslog-ng PE application. The **syslog-ng-ctl stop** works like a SIGHUP (-15) on Linux and Unix systems. On Microsoft Windows, this is the only way to gracefully stop syslog-ng PE if it is running in the foreground.

Files

`/opt/syslog-ng/sbin/syslog-ng-ctl`

See also

The syslog-ng Documentation page

syslog-ng.conf(5)

syslog-ng(8)

Note

For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**

If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.

For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The syslog-ng manual page

Table of Contents

Name

syslog-ng — syslog-ng system logger application

Synopsis

`syslog-ng` [options]

Description

This manual page is only an abstract, for the complete documentation of syslog-ng, see **The syslog-ng Premium Edition Administrator Guide** or the syslog-ng Documentation page.

The syslog-ng PE application is a flexible and highly scalable system logging application. Typically, syslog-ng is used to manage log messages and implement centralized logging, where the aim is to collect the log messages of several devices on a single, central log server. The different devices - called syslog-ng clients - all run syslog-ng, and collect the log messages from the various applications, files, and other *sources*. The clients send all

important log messages to the remote syslog-ng server, where the server sorts and stores them.

Options

--caps

Run syslog-ng PE process with the specified POSIX capability flags.

- If the *--no-caps* option is not set, and the host supports CAP_SYSLOG, syslog-ng PE uses the following capabilities: "cap_net_bind_service, cap_net_broadcast, cap_net_raw, cap_dac_read_search, cap_dac_override, cap_chown, cap_fowner=p cap_syslog=ep"
- If the *--no-caps* option is not set, and the host does not support CAP_SYSLOG, syslog-ng PE uses the following capabilities: "cap_net_bind_service, cap_net_broadcast, cap_net_raw,cap_dac_read_search, cap_dac_override, cap_chown, cap_fowner=p cap_sys_admin=ep"

For example:

```
             /opt/syslog-ng/sbin/syslog-ng -Fv --caps cap_sys_admin,cap_
chown,cap_dac_override,cap_net_bind_service,cap_fowner=pi
```

Note that the capabilities are not case sensitive, the following command is also good: **/opt/syslog-ng/sbin/syslog-ng -Fv --caps CAP_SYS_ADMIN,CAP_CHOWN,CAP_DAC_OVERRIDE,CAP_NET_BIND_SERVICE,CAP_FOWNER=pi**

For details on the capability flags, see the following man pages: `cap_from_text(3)` and `capabilities(7)`

--cfgfile <file> or **-f <file>**

Use the specified configuration file.

--chroot <dir> or **-C <dir>**

Change root to the specified directory. The configuration file is read after chrooting so, the configuration file must be available within the chroot. That way it is also possible to reload the syslog-ng configuration after chrooting. However, note that the *--user* and *--group* options are resolved before chrooting.

--control <file> or **-c <file>**

Set the location of the syslog-ng control socket. Default value: `/var/run/syslog-ng.ctl`

--debug or **-d**

Start syslog-ng in debug mode.

--default-modules

A comma-separated list of the modules that are loaded automatically. Modules not loaded automatically can be loaded by including the **`@module <modulename>`** statement in the syslog-ng PE configuration file. Available only in syslog-ng Premium

Edition 4.1 and later.

--enable-core

Enable syslog-ng to write core files in case of a crash to help support and debugging.

--fd-limit <number>

Set the minimal number of required file descriptors (fd-s). This sets how many files syslog-ng can keep open simultaneously. Default value: `4096`. Note that this does not override the global ulimit setting of the host.

--foreground or **-F**

Do not daemonize, run in the foreground. When running in the foreground, syslog-ng PE starts from the current directory (**$CWD**) so it can create core files (normally, syslog-ng PE starts from `$PREFIX/var`).

--group <group> or **-g <group>**

Switch to the specified group after initializing the configuration file.

--help or **-h**

Display a brief help message.

--module-registry

Display the list and description of the available modules. Note that not all of these modules are loaded automatically, only the ones specified in the **--default-modules** option. Available only in syslog-ng Premium Edition 4 F1 and later.

--no-caps

Run syslog-ng as root, without capability-support. This is the default behavior. On Linux, it is possible to run syslog-ng as non-root with capability-support if syslog-ng was compiled with the `--enable-linux-caps` option enabled. (Execute **syslog-ng --version** to display the list of enabled build parameters.)

To run syslog-ng PE with specific capabilities, use the `--caps` option.

--persist-file <persist-file> or **-R <persist-file>**

Set the path and name of the `syslog-ng.persist` file where the persistent options and data are stored.

--pidfile <pidfile> or **-p <pidfile>**

Set path to the PID file where the pid of the main process is stored.

--preprocess-into <output-file>

After processing the configuration file and resolving included files and variables, write the resulting configuration into the specified output file. Available only in syslog-ng Premium Edition 4 F1 and later.

--process-mode <mode>

Sets how to run syslog-ng: in the `foreground` (mainly used for debugging), in the `background` as a daemon, or in `safe-background` mode. By default, syslog-ng runs in `safe-background` mode. This mode creates a supervisor process called `supervising syslog-ng`, that restarts syslog-ng if it crashes.

--stderr or **-e**

Log internal messages of syslog-ng to stderr. Mainly used for debugging purposes in conjunction with the `--foreground` option. If not specified, syslog-ng will log such messages to its internal source.

--syntax-only or **-s**

Verify that the configuration file is syntactically correct and exit.

--user <user> or **-u <user>**

Switch to the specified user after initializing the configuration file (and optionally chrooting). Note that it is not possible to reload the syslog-ng configuration if the specified user has no privilege to create the `/dev/log` file.

--verbose or **-v**

Enable verbose logging used to troubleshoot syslog-ng.

--version or **-V**

Display version number and compilation information, and also the list and short description of the available modules. For detailed description of the available modules, see the **--module-registry** option. Note that not all of these modules are loaded automatically, only the ones specified in the **--default-modules** option.

--worker-threads

Sets the number of worker threads syslog-ng PE can use, including the main syslog-ng PE thread. Note that certain operations in syslog-ng PE can use threads that are not limited by this option. This setting has effect only when syslog-ng PE is running in multithreaded mode. Available only in syslog-ng Premium Edition 4 F1 and later. See **The syslog-ng Premium Edition 7 Administrator Guide** for details.

Setting default command-line options

You can set default settings for syslog-ng PE — syslog-ng PE will always run with these default command-line parameters. You can specify your default settings in the following files:

- `/etc/default/syslog-ng`
- `/etc/sysconfig/syslog-ng` (only for RedHat platforms)
- `$SYSLOGNG_PREFIX/etc/default/syslog-ng`, where $SYSLOGNG_PREFIX is the installation directory of syslog-ng PE. For version 4.0, this is `/opt/syslog-ng`

During startup, syslog-ng PE will automatically use the settings from these files if they exist. You can set the following options:

MAXWAIT

The number of seconds the init script will wait for syslog-ng PE to shut down properly. If the syslog-ng PE process does not shut down during this period, it is terminated with a SIGKILL signal. Increase this value if you have lots of separate disk-buffer files (for example, to 60 seconds).

SYSLOGNG_OPTIONS

A string of additional command-line options for the syslog-ng daemon.

Files

`/opt/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng.conf`

See also

syslog-ng.conf(5)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**
>
> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

The syslog-ng.conf manual page

Table of Contents

Name

syslog-ng.conf — syslog-ng configuration file

Synopsis

`syslog-ng.conf`

Description

This manual page is only an abstract, for the complete documentation of syslog-ng, see **The syslog-ng Premium Edition Administrator Guide** or the syslog-ng Documentation page.

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

Basic concepts of syslog-ng PE

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.

Configuring syslog-ng

- The main body of the configuration file consists of object definitions: sources, destinations, logpaths 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. Object definitions (also called statements) have the following syntax:

```
type-of-the-object identifier-of-the-object {<parameters>};
```

- *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.

  All identifiers, attributes, and any other strings used in the syslog-ng configuration file are case sensitive.

  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.

  Note

  Repeating a definition of an object (that is, defining the same object with the same id more than once) is not allowed, unless you use the `@define allow-config-dups 1` definition in the configuration file.
- *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 1. 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 ???.

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

- Objects can be used before definition.
- Objects can be defined inline as well. This is useful if you use the object only once (for example, a filter). For details, see ???.
- 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)); };
  ```

The syntax of log statements is as follows:

```
log {
      source(s1); source(s2); ...
      optional_element(filter1|parser1|rewrite1);
      optional_element(filter2|parser2|rewrite2);
      ...
      destination(d1); destination(d2); ...
      flags(flag1[, flag2...]);
};
```

The following log statement sends all messages arriving to the localhost to a remote server.

```
        source s_localhost { 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); };
```

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 2. Using global options

To disable domain name resolving, add the following line to the syslog-ng configuration file:

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

The sources, destinations, and filters available in syslog-ng are listed below. For details, see the syslog-ng Documentation page.

Table 1. 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, 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.
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.

Name	Description
windowsevent()	Reads messages from the Windows Event Collector tool.

Table 2. Destination drivers available in syslog-ng

Name	Description
elasticsearch2	Sends messages to an Elasticsearch server. The *elasticsearch2* driver supports Elasticsearch version 2 and newer.
file()	Writes messages to the specified file.
hdfs()	Sends messages into a file on a Hadoop Distributed File System (HDFS) or MapR-FS node.
http()	Sends messages over the HTTP protocol.
kafka()	Publishes log messages to the Apache Kafka message bus, where subscribers can access them.
logstore()	Writes messages securely into encrypted, compressed, and timestamped binary files.
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, 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 email messages to the specified recipients.
Splunk	Forward your log messages to Splunk.
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 ???.
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.

Table 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.

Name	Description
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.

Files

`/opt/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng.conf`

See also

syslog-ng(8)

> **Note**
>
> For the detailed documentation of syslog-ng PE see **The syslog-ng PE 7 Administrator Guide**
>
> If you experience any problems or need help with syslog-ng, visit the **syslog-ng mailing list**.
>
> For news and notifications about of syslog-ng, visit the **syslog-ng blogs**.

Author

This manual page was written by the One Identity Documentation Team.

Copyright

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

A

alias IP
An additional IP address assigned to an interface that already has an IP address. The normal and alias IP addresses both refer to the same physical interface.

auditing policy
The auditing policy determines which events are logged on host running Microsoft Windows operating systems.

authentication
The process of verifying the authenticity of a user or client before allowing access to a network system or service.

B

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

BSD-syslog protocol
The old syslog protocol standard described in RFC 3164. Sometimes also referred to as the legacy-syslog protocol.

C

CA
A Certificate Authority (CA) is an institute that issues certificates.

Cadence
[[[Undefined variable TemplateGuideVariables.OneIdentityNameShort]]] font that contains standard icons used in the user interfaces for various [[[Undefined variable TemplateGuideVariables.OneIdentityNameShort]]] products.

certificate
A certificate is a file that uniquely identifies its owner. Certificates contains information identifying the owner of the certificate, a public key itself, the expiration date of the certificate, the name of the CA that signed the certificate, and some other data.

client mode
In client mode, syslog-ng collects the local logs generated by the host and forwards them through a network connection to the central syslog-ng server or to a relay.

D

destination
A named collection of configured destination drivers.

destination driver
A communication method used to send log messages.

destination, local
A destination that transfers log messages within the host, for example, writes them to a file, or passes them to a log analyzing application.

destination, network
A destination that sends log messages to a remote host (that is, a syslog-ng relay or server) using a network connection.

disk buffer
The Premium Edition of syslog-ng can store messages on the local hard disk if the central log server or the network connection to the server becomes unavailable.

disk queue
See disk buffer.

domain name
The name of a network, for example: balabit.com.

Drop-down
Flare default style, that can be used to group content within a topic. It is a resource to structure and collapse content especially in non-print outputs.

E

embedded log statement
A log statement that is included in another log statement to create a complex log path.

F

filter
An expression to select messages.

fully qualified domain name (FQDN)
A domain name that specifies its exact location in the tree hierarchy of the Domain Name System (DNS). For example, given a device with a local hostname myhost and a parent domain name example.com, the fully qualified domain name is myhost.example.com.

G

gateway
A device that connects two or more parts of the network, for example: your local intranet and the external network (the Internet). Gateways act as entrances into other networks.

Glossary
List of short definitions of product specific terms.

H

high availability
High availability uses a second syslog-ng server unit to ensure that the logs are received even if the first unit breaks down.

host
A computer connected to the network.

hostname
A name that identifies a host on the network.

I

IETF-syslog protocol
The syslog-protocol standard developed by the Internet Engineering Task Force (IETF), described in RFC 5424-5427.

K

key pair
A private key and its related public key. The private key is known only to the owner, while the public key can be freely distributed. Information encrypted with the private key can only be decrypted using the public key.

L

license
The syslog-ng license determines the number of distinct hosts (clients and relays) that can connect to the syslog-ng server.

log path
A combination of sources, filters, parsers, rewrite rules, and destinations: syslog-ng examines all messages arriving to the sources of the logpath and sends the messages matching all filters to the defined destinations.

log source host
A host or network device (including syslog-ng clients and relays) that sends logs to the syslog-ng server. Log source hosts can be servers, routers, desktop computers, or other devices capable of sending syslog messages or running syslog-ng.

log statement
See log path.

logstore
A binary logfile format that can encrypt, compress, and timestamp log messages.

Long Term Supported release
Long Term Supported releases are major releases of that are supported for three years after their original release.

LSH
See log source host.

N

name server
A network computer storing the IP addresses corresponding to domain names.

Note
Circumstance, that needs special attention.

O

Oracle Instant Client
The Oracle Instant Client is a small set of libraries, which allow you to connect to an Oracle Database. A subset of the full Oracle Client, it requires minimal installation but has full functionality.

output buffer
A part of the memory of the host where syslog-ng stores outgoing log messages if the destination cannot accept the messages immediately.

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

overflow queue
See output buffer.

P

parser
A set of rules to segment messages into named fields or columns.

ping
A command that sends a message from a host to another host over a network to test connectivity and packet loss.

port
A number ranging from 1 to 65535 that identifies the destination application of the transmitted data. For example: SSH commonly uses port 22, web servers (HTTP) use port 80, and so on.

Public-key authentication
An authentication method that uses encryption key pairs to verify the identity of a user or a client.

R

regular expression
A regular expression is a string that describes or matches a set of strings.

relay mode
In relay mode, syslog-ng receives logs through the network from syslog-ng clients and forwards them to the central syslog-ng server using a network connection.

rewrite rule
A set of rules to modify selected elements of a log message.

S

SaaS
Software-as-a-Service.

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

Skin
Used to design the online output window.

Snippet
Flare file type that can be used to reuse content. The One Identity syslog-ng PE contains various default snippets.

source
A named collection of configured source drivers.

source driver
A communication method used to receive log messages.

source, local
A source that receives log messages from within the host, for example, from a file.

source, network
A source that receives log messages from a remote host using a network connection, for example, network(), syslog().

SSL
See TLS.

syslog-ng
The syslog-ng application is a flexible and highly scalable system logging application, typically used to manage log messages and implement centralized logging.

syslog-ng agent
The syslog-ng Agent for Windows is a commercial log collector and forwarder application for the Microsoft Windows platform. It collects the log messages of the Windows-based host and forwards them to a syslog-ng server using regular or SSL-encrypted TCP connections.

syslog-ng client
A host running syslog-ng in client mode.

syslog-ng Premium Edition
The syslog-ng Premium Edition is the commercial version of the open-source application. It offers additional features, like encrypted message transfer and an agent for Microsoft Windows platforms.

syslog-ng relay
A host running syslog-ng in relay mode.

syslog-ng server
A host running syslog-ng in server mode.

T

template
A user-defined structure that can be used to restructure log messages or automatically generate file names.

Tip
Additional, usefull information.

TLS
Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols which provide secure communications on the Internet. The application can encrypt the communication between the clients and the server using TLS to prevent unauthorized access to sensitive log messages.

traceroute
A command that shows all routing steps (the path of a message) between two hosts.

U

UNIX domain socket

A UNIX domain socket (UDS) or IPC socket (inter-procedure call socket) is a virtual socket, used for inter-process communication.