syslog-ng Open Source Edition 3.17

Administration Guide

Copyright 2018 One Identity LLC.

ALL RIGHTS RESERVED.

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of One Identity LLC .

The information in this document is provided in connection with One Identity products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of One Identity LLC products. EXCEPT AS SET FORTH IN THE TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, ONE IDENTITY ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ONE IDENTITY BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ONE IDENTITY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. One Identity makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. One Identity does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

One Identity LLC.
Attn: LEGAL Dept
4 Polaris Way
Aliso Viejo, CA 92656

Refer to our Web site (http://www.OneIdentity.com) for regional and international office information.

Patents

One Identity is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at http://www.OneIdentity.com/legal/patents.aspx.

Trademarks

One Identity and the One Identity logo are trademarks and registered trademarks of One Identity LLC. in the U.S.A. and other countries. For a complete list of One Identity trademarks, please visit our website at www.OneIdentity.com/legal. All other trademarks are the property of their respective owners.

Legend

WARNING: A WARNING icon indicates a potential for property damage, personal injury, or death.

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

IMPORTANT, NOTE, TIP, MOBILE, or VIDEO: An information icon indicates supporting information.

syslog-ng OSE Administration Guide
Updated - August 2018
Version - 3.17

Contents

Preface **18**
Summary of contents 18
Target audience and prerequisites 19
Products covered in this guide 19
Summary of changes 20
Version 3.16 - 3.17 20
Version 3.15 - 3.16 21
Version 3.14 - 3.15 21
Version 3.13 - 3.14 21
Version 3.12 - 3.13 22
Version 3.11 - 3.12 23
Version 3.10 - 3.11 24
Version 3.9 - 3.10 24
Version 3.8 - 3.9 25
Version 3.7 - 3.8 26
Version 3.6 - 3.7 27
Version 3.5 - 3.6 28
Feedback 29
Acknowledgments 30

Introduction to syslog-ng **31**
What syslog-ng is 31
Secure and reliable log transfer 31
Flexible data extraction and processing 32
Big data clusters 32
Message queue support 32
SQL, NoSQL, and monitoring 33
Wide protocol and platform support 33
What syslog-ng is not 33
Why is syslog-ng needed? 33
What is new in syslog-ng Open Source Edition 3.17? 34
Who uses syslog-ng? 35

Supported platforms 35
The concepts of syslog-ng 36
The philosophy of syslog-ng 36
Logging with syslog-ng 36
The route of a log message in syslog-ng 37
Modes of operation 38
Client mode 38
Relay mode 39
Server mode 39
Global objects 40
Timezones and daylight saving 41
How syslog-ng OSE assigns timezone to the message 42
A note on timezones and timestamps 43
Product licensing 43
High availability support 43
The structure of a log message 43
BSD-syslog or legacy-syslog messages 44
The PRI message part 44
The HEADER message part 46
The MSG message part 46
IETF-syslog messages 46
The PRI message part 47
The HEADER message part 48
The STRUCTURED-DATA message part 49
The MSG message part 49
Enterprise-wide message model (EWMM) 49
Message representation in syslog-ng OSE 50
Structuring macros, metadata, and other value-pairs 52
Specifying data types in value-pairs 53
value-pairs() 54
Things to consider when forwarding messages between syslog-ng OSE hosts 58
Commercial version of syslog-ng 60
Installing syslog-ng 63
Compiling syslog-ng from source 63

Compiling options of syslog-ng OSE 65
Uninstalling syslog-ng OSE 68
Configuring Microsoft SQL Server to accept logs from syslog-ng 68

The syslog-ng OSE quick-start guide 75
Configuring syslog-ng on client hosts 75
Configuring syslog-ng on server hosts 78
Configuring syslog-ng relays 80
Configuring syslog-ng on relay hosts 80
How relaying log messages works 82

The syslog-ng OSE configuration file 84
Location of the syslog-ng configuration file 84
The configuration syntax in detail 84
Notes about the configuration syntax 87
Defining configuration objects inline 88
Using channels in configuration objects 89
Global and environmental variables 91
Modules in syslog-ng OSE 92
Loading modules 92
Managing complex syslog-ng configurations 93
Including configuration files 93
Reusing configuration blocks 94
Mandatory parameters 96
Passing arguments to configuration blocks 97
Generating configuration blocks from a script 98

source: Read, receive, and collect log messages 101
How sources work 101
default-network-drivers: Receive and parse common syslog messages 105
default-network-drivers() source options 107
internal: Collecting internal messages 110
internal() source options 110
file: Collecting messages from text files 112
Notes on reading kernel messages 113
file() source options 113
wildcard-file: Collecting messages from multiple text files 124

wildcard-file() source options ... 125
linux-audit: Collecting messages from Linux audit logs ... 138
linux-audit() source options ... 139
network: Collecting messages using the RFC3164 protocol (network() driver) ... 140
network() source options ... 141
nodejs: Receiving JSON messages from nodejs applications ... 153
nodejs() source options ... 154
mbox: Converting local e-mail messages to log messages ... 156
mbox() source options ... 157
osquery: Collect and parse osquery result logs ... 158
osquery() source options ... 161
pipe: Collecting messages from named pipes ... 164
pipe() source options ... 164
pacct: Collecting process accounting logs on Linux ... 175
pacct() options ... 175
program: Receiving messages from external applications ... 177
program() source options ... 178
snmptrap: Read Net-SNMP traps ... 185
snmptrap() source options ... 188
sun-streams: Collecting messages on Sun Solaris ... 191
sun-streams() source options ... 191
syslog: Collecting messages using the IETF syslog protocol (syslog() driver) ... 198
syslog() source options ... 199
system: Collecting the system-specific log messages of a platform ... 211
system() source options ... 213
systemd-journal: Collecting messages from the systemd-journal system log storage 215
systemd-journal() source options ... 218
systemd-syslog: Collecting systemd messages using a socket ... 222
systemd-syslog() source options ... 223
tcp, tcp6, udp, udp6: Collecting messages from remote hosts using the BSD syslog protocol— OBSOLETE ... 224
tcp(), tcp6(), udp() and udp6() source options: OBSOLETE ... 225
Change an old source driver to the network() driver ... 225
unix-stream, unix-dgram: Collecting messages from UNIX domain sockets ... 226
UNIX credentials and other metadata ... 227

unix-stream() and unix-dgram() source options ... 228
stdin: Collecting messages from the standard input stream ... 236
stdin() source options ... 237
destination: Forward, send, and store log messages ... 248
amqp: Publishing messages using AMQP ... 250
amqp() destination options ... 251
elasticsearch: Sending messages directly to Elasticsearch version 1.x ... 261
Prerequisites ... 263
How syslog-ng OSE interacts with Elasticsearch ... 264
Client modes ... 265
Elasticsearch destination options ... 265
elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher ... 277
Prerequisites ... 280
How syslog-ng OSE interacts with Elasticsearch ... 280
Client modes ... 281
Search Guard and syslog-ng OSE ... 282
Elasticsearch2 destination options ... 284
Example use cases of sending logs to Elasticsearch using syslog-ng ... 304
file: Storing messages in plain-text files ... 304
file() destination options ... 305
graphite: Sending metrics to Graphite ... 316
graphite() destination options ... 317
Sending logs to Graylog ... 320
graylog2() destination options ... 321
hdfs: Storing messages on the Hadoop Distributed File System (HDFS) ... 323
Prerequisites ... 324
How syslog-ng OSE interacts with HDFS ... 325
Storing messages with MapR-FS ... 326
Kerberos authentication with syslog-ng hdfs() destination ... 327
HDFS destination options ... 328
Posting messages over HTTP ... 339
HTTP destination options ... 340
http: Posting messages over HTTP without Java ... 344
HTTP destination options ... 345
kafka: Publishing messages to Apache Kafka ... 359

Prerequisites ... 360
How syslog-ng OSE interacts with Apache Kafka ... 361
Kafka destination options ... 361
loggly: Using Loggly ... 368
loggly() destination options ... 369
logmatic: Using Logmatic.io ... 370
logmatic() destination options ... 372
mongodb: Storing messages in a MongoDB database ... 374
How syslog-ng OSE connects the MongoDB server ... 375
mongodb() destination options ... 376
network: Sending messages to a remote log server using the RFC3164 protocol (network() driver) ... 385
network() destination options ... 386
osquery: Sending log messages to osquery's syslog table ... 402
osquery() destination options ... 403
pipe: Sending messages to named pipes ... 405
pipe() destination options ... 406
program: Sending messages to external applications ... 413
program() destination options ... 414
pseudofile() ... 423
pseudofile() destination options ... 423
redis: Storing name-value pairs in Redis ... 425
redis() destination options ... 426
riemann: Monitoring your data with Riemann ... 432
riemann() destination options ... 433
smtp: Generating SMTP messages (e-mail) from logs ... 445
smtp() destination options ... 446
Splunk: Sending log messages to Splunk ... 454
sql: Storing messages in an SQL database ... 454
Using the sql() driver with an Oracle database ... 456
Using the sql() driver with a Microsoft SQL database ... 457
The way syslog-ng interacts with the database ... 459
MySQL-specific interaction methods ... 460
MsSQL-specific interaction methods ... 460
sql() destination options ... 460

stomp: Publishing messages using STOMP ... 472
stomp() destination options ... 473
syslog: Sending messages to a remote logserver using the IETF-syslog protocol ... 479
syslog() destination options ... 480
syslog-ng() destination options ... 496
tcp, tcp6, udp, udp6: Sending messages to a remote log server using the legacy BSD-syslog protocol (tcp(), udp() drivers) ... 510
tcp(), tcp6(), udp(), and udp6() destination options ... 510
Change an old destination driver to the network() driver ... 511
Telegram: Sending messages to Telegram ... 512
telegram() destination options ... 512
unix-stream, unix-dgram: Sending messages to UNIX domain sockets ... 515
unix-stream() and unix-dgram() destination options ... 516
usertty: Sending messages to a user terminal: usertty() destination ... 525
Write your own custom destination in Java or Python ... 526
Client-side failover ... 526

log: Filter and route log messages using log paths, flags, and filters ... 529

Log paths ... 529
Embedded log statements ... 530
Using embedded log statements ... 532
if-else-elif: Conditional expressions ... 534
Junctions and channels ... 534
Log path flags ... 537
Managing incoming and outgoing messages with flow-control ... 540
Flow-control and multiple destinations ... 544
Configuring flow-control ... 544
Using disk-based and memory buffering ... 546
Enabling reliable disk-based buffering ... 548
Enabling normal disk-based buffering ... 549
Enabling memory buffering ... 549
About disk queue files ... 550
Filters ... 551
Using filters ... 551
Combining filters with boolean operators ... 552
Comparing macro values in filters ... 553

Using wildcards, special characters, and regular expressions in filters 554
Tagging messages 555
Filter functions 556
Dropping messages 561

Global options of syslog-ng OSE 563
Configuring global syslog-ng options 563
Global options 563

TLS-encrypted message transfer 581
Secure logging using TLS 581
Encrypting log messages with TLS 582
Configuring TLS on the syslog-ng clients 583
Configuring TLS on the syslog-ng server 584
Mutual authentication using TLS 586
Configuring TLS on the syslog-ng clients 587
Configuring TLS on the syslog-ng server 588
Password-protected keys 590
TLS options 591

template and rewrite: Format, modify, and manipulate log messages 598
Customize message format using macros and templates 598
Formatting messages, filenames, directories, and tablenames 599
Templates and macros 599
Date-related macros 601
Hard vs. soft macros 602
Macros of syslog-ng OSE 603
Using template functions 612
Template functions of syslog-ng OSE 613
Modifying the on-the-wire message format 635
Modifying messages using rewrite rules 635
Replacing message parts 636
Setting message fields to specific values 637
Unsetting message fields 640
Creating custom SDATA fields 641
Setting multiple message fields to specific values 642
map-value-pairs: Rename value-pairs to normalize logs 643

Conditional rewrites 644
How conditional rewriting works 644
Adding and deleting tags 645
Anonymizing credit card numbers 646
Regular expressions 646
Types and options of regular expressions 647
Optimizing regular expressions 649
parser: Parse and segment structured messages 651
Parsing syslog messages 652
Options of syslog-parser parsers 654
Parsing messages with comma-separated and similar values 656
Options of CSV parsers 659
Parsing key=value pairs 663
Options of key=value parsers 666
The JSON parser 667
The JSON parser The JSON parser 667
Options of JSON parsers 670
The XML parser 672
Options of XML parsers 675
Parsing dates and timestamps 678
Options of date-parser() parsers 679
The Apache Access Log Parser 681
Options of apache-accesslog-parser() parsers 682
The Cisco Parser 683
The Linux Audit Parser 685
Options of linux-audit-parser() parsers 687
The Python Parser 688
Parsing enterprise-wide message model (EWMM) messages 693
The sudo parser 693
The iptables parser 694
db-parser: Process message content with a pattern database (patterndb) 696
Classifying log messages 696
The structure of the pattern database 697
How pattern matching works 698

Artificial ignorance 699
Using pattern databases 700
Using parser results in filters and templates 701
Downloading sample pattern databases 703
Correlating log messages using pattern databases 704
Referencing earlier messages of the context 706
Triggering actions for identified messages 707
Conditional actions 709
External actions 710
Actions and message correlation 711
Creating pattern databases 714
Using pattern parsers 714
Pattern parsers of syslog-ng OSE 716
What's new in the syslog-ng pattern database format V5 719
The syslog-ng pattern database format 719
Element: patterndb 721
Element: ruleset 721
Element: patterns 722
Element: rules 723
Element: rule 724
Element: patterns 726
Element: urls 727
Element: values 728
Element: examples 728
Element: example 729
Element: actions 730
Element: action 732
Element: create-context 734
Element: tags 737

Correlating log messages 738

Correlating messages using the grouping-by() parser 738
Referencing earlier messages of the context 742
Options of grouping-by parsers 743

Enriching log messages with external data 747

Adding metadata from an external file 747
Using filters as selector 749
Options add-contextual-data() 750
Looking up GeoIP data from IP addresses (DEPRECATED) 752
Options of geoip parsers 754
Looking up GeoIP2 data from IP addresses 755
Referring to parts of the message as a macro 756
Using the GeoIP2 parser 756
Transferring your logs to Elasticsearch using GeoIP2 757
Options of geoip2 parsers 758
Statistics of syslog-ng 760
Metrics and counters of syslog-ng OSE 760
Log statistics from the internal() source 763
Multithreading and scaling in syslog-ng OSE 765
Multithreading concepts of syslog-ng OSE 765
Configuring multithreading 767
Optimizing multithreaded performance 767
Troubleshooting syslog-ng 769
Possible causes of losing log messages 770
Creating syslog-ng core files 771
Collecting debugging information with strace, truss, or tusc 771
Running a failure script 772
Stopping syslog-ng 773
Reporting bugs and finding help 774
Recover data from orphaned diskbuffer files 774
No local logs after specifying an unusual storage directory 774
No logs after specifying an unusual port number 774
Error messages 775
Best practices and examples 777
General recommendations 777
Handling large message load 777
Using name resolution in syslog-ng 778
Resolving hostnames locally 779
Collecting logs from chroot 779

Configuring log rotation 780

The syslog-ng manual pages 782

The dqtool tool manual page 782

Name 782

Synopsis 782

Description 782

The cat command 783

Files 784

See also 784

Author 784

Copyright 784

The loggen manual page 784

Name 785

Synopsis 785

Description 785

Options 785

Examples 788

Files 788

See also 788

Author 789

Copyright 789

The pdbtool manual page 789

Name 789

Synopsis 789

Description 790

The dictionary command 790

The dump command 790

The match command 791

The merge command 793

The patternize command 794

The test command 795

Files 795

See also 795

Author 796

Copyright 796

The syslog-ng control tool manual page 796
Name 796
Synopsis 796
Description 797
Enabling troubleshooting messages 797
syslog-ng-ctl query 798
The stats command 800
Handling password-protected private keys 801
Reloading the configuration 802
Files 802
See also 803
Author 803
Copyright 803
The syslog-ng-debun manual page 803
Name 803
Synopsis 804
Description 804
General Options 804
Debug mode options 804
System call tracing 805
Packet capture options 805
Examples 805
Files 807
See also 807
Author 807
Copyright 807
The syslog-ng manual page 807
Name 807
Synopsis 808
Description 808
Options 808
Files 811
See also 811
Author 811
Copyright 811

The syslog-ng.conf manual page ... 811
Name ... 812
Synopsis ... 812
Description ... 812
Basic concepts of syslog-ng OSE ... 812
Configuring syslog-ng ... 813
Files ... 817
See also ... 817
Author ... 818
Copyright ... 818
Third-party contributions ... 819
GNU General Public License ... 819
Preamble ... 819
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION ... 820
Section 0 ... 820
Section 1 ... 821
Section 2 ... 821
Section 3 ... 822
Section 4 ... 822
Section 5 ... 822
Section 6 ... 823
Section 7 ... 823
Section 8 ... 823
Section 9 ... 824
Section 10 ... 824
NO WARRANTY Section 11 ... 824
Section 12 ... 824
How to Apply These Terms to Your New Programs ... 825
GNU Lesser General Public License ... 826
Preamble ... 826
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION ... 828
Section 0 ... 828
Section 1 ... 828
Section 2 ... 829
Section 3 ... 829

Section 4 830
Section 5 830
Section 6 831
Section 7 832
Section 8 832
Section 9 832
Section 10 832
Section 11 833
Section 12 833
Section 13 833
Section 14 834
NO WARRANTY Section 15 834
NO WARRANTY Section 16 834
How to Apply These Terms to Your New Libraries 834
License attributions 835

Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License 836

About us 842
Contacting us 842
Technical support resources 842

Preface

Welcome to the syslog-ng Open Source Edition 3.17 Administrator Guide!

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

Summary of contents

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

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

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

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

The syslog-ng OSE 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.

source: Read, receive, and collect log messages explains how to collect and receive log messages from various sources.

destination: Forward, send, and store log messages describes the different methods to store and forward log messages.

log: Filter and route log messages using 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 OSE lists the global options of syslog-ng OSE and explains how to use them.

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

template and rewrite: Format, modify, and manipulate log 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.

db-parser: Process message content with a pattern database (patterndb) explains how to identify and process log messages using a pattern database.

Correlating log messages explains how to correlate log messages that match a set of filters or that are identified using a pattern database.

Enriching log messages with external data explains 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.

Statistics of syslog-ng details the available statistics that syslog-ng OSE collects about the processed log messages.

Multithreading and scaling in syslog-ng OSE describes how to configure syslog-ng OSE 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 OSE.

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

Third-party contributions includes the text of the licenses applicable to syslog-ng Open Source Edition.

Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License includes the text of the Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License applicable to The syslog-ng Open Source Edition 3.17 Administrator Guide.

Target audience and prerequisites

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

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

- At least basic system administration knowledge.
- An understanding of networks, TCP/IP protocols, and general network terminology.
- Working knowledge of the UNIX or Linux operating system.
- In-depth knowledge of the logging process of various platforms and applications.
- An understanding of the legacy syslog (BSD-syslog) protocol and the new syslog (IETF-syslog) protocol standard.

Products covered in this guide

This guide describes the use of the following products:

- syslog-ng Open Source Edition (syslog-ng OSE) 3.17.1 and later

Summary of changes

This section lists the changes of The syslog-ng Open Source Edition Administrator Guide.

Version 3.16 - 3.17

Changes in product:

- A new source driver, `linux-audit()`, has been added. The `linux-audit()` source reads and automatically parses the Linux audit logs. For details, see linux-audit: Collecting messages from Linux audit logs.
- A new system source option, exclude-kmsg() makes it possible to avoid duplicate collection of kernel logs or errors in kernel log collection (for example, in scenarios where the log management on the host system and the containerized solution are collecting the kernel logs simultaneously). When set to `yes`, syslog-ng OSE will omit kernel logs on platforms where they are available separately.
- You can now refer to any additional parameters at the end of the argument in a block by adding three dots to it (…). It tells syslog-ng OSE that this macro accepts `__VARARGS__`, therefore any name-value pair can be passed without validation. For details, see Passing arguments to configuration blocks.
- You can now make parameters mandatory in block definitions by defining them with empty brackets `()`. For details, see Mandatory parameters.
- The `failover()` option allows you to specify what happens after syslog-ng OSE fails over to a secondary server. Additionally, the `failover-servers()` option has been deprecated and removed from the document. For more information about the `failover()` option, see Client-side failover on page 526.
- Added support for the timestamp format used by Cisco Unified Call Manager in the Cisco Parser. For details, see the source code of this parser on GitHub.

Changes in documentation:

- A note about JVM still running after deleting all Java destinations and reloading syslog-ng has been added to the description of Java destinations.
- The default value of the `--skip-tokens` parameter of the loggen application has been changed to `0`. For details, see The loggen manual page.

Version 3.15 - 3.16

Changes in product:

- A new destination driver, `telegram()`, has been added. The `telegram()` destination sends log messages to Telegram, which is a secure, cloud-based mobile and desktop messaging app. For more information, see Telegram: Sending messages to Telegram.
- A new template function, `urlencode`, has been added. You can use the `urlencode` template function together with the `telegram()` destination to send syslog messages to Telegram. For more information, see urlencode.
- To ensure that a module is loaded, you can use the `@requires` statement. For more information, see Loading modules.
- The `add-contextual-data()` has been extended with the `ignore-case()` option. For more information, see Options add-contextual-data().
- The `hook-commands()` has been added, which makes it possible to execute external programs when they are initialized or torn down. The `hook-commands()` can be used for both source and destination drivers. For more information, see hook-commands().

Version 3.14 - 3.15

Changes in product:

- It is now possible to use `if {}`, `elif {}`, and `else {}` blocks to configure conditional expressions. For details, see if-else-elif: Conditional expressions.
- A new log path flag, `drop-unmatched`, has been added. The new flag causes messages to be dropped along a log path when they do not match a filter or are discarded by a parser. For details, see Log path flags.
- Support for Elasticsearch's Shield has been removed.
- Support for POSIX regular expressions has been removed.

Version 3.13 - 3.14

Changes in product:

- You can use password-protected private keys in the `network()` and `syslog()` source and destination drivers. For details, see Password-protected keys.
- 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 OSE evaluates the filters in the order they appear in the database file. If a filter matches the message, syslog-ng OSE adds the name-value pair related to the filter. For details, see Using filters as selector.

Version 3.12 - 3.13

Changes in product:

- A new source driver, `stdin()`, has been added. The `stdin()` driver collects messages from the standard input stream. For more information, see stdin: Collecting messages from the standard input stream.
- A new destination, Sending logs to Graylog, and a template to send syslog messages to Graylog, format-gelf, has been added.
- A new template function, `getent`, has been added. You can use the `getent` template function to look up entries from the Name Service Switch libraries. For more information, see getent.
- The default values of the `--enable-json`, `--enable-mongodb`, and `--with-libmongo-client` compile parameters have changed. For more information, see Compiling options of syslog-ng OSE.
- A new compile option, *--with-module-path*, has been added. The new option specifies syslog-ng OSE's module installation directory. For more information, see Compiling options of syslog-ng OSE.
- A new destination driver, `osquery()`, has been added. The new driver sends log messages to osquery's syslog table. For more information, see osquery: Sending log messages to osquery's syslog table.
- It is now possible to specify TLS options in a `tls()` block. For more information, see:
  - amqp() destination options
  - HTTP destination options
  - riemann() destination options
- Support for microseconds in Riemann destinations has been introduced. For more information, see event-time().
- Module auto-loading now also works for the `system()` source. For more information, see --default-modules .

Changes in documentation:

- A new section describing common error messages has been added to the document. For more information, see Error messages .
- Several corrections and editorial changes.

Version 3.11 - 3.12

Changes in product:

- A new `systemd-journal()` source option, called `read-old-records()`, has been added. For more information, see read-old-records().
- An option called `jvm-options()` has been added, which allows you to fine-tune Java Virtual Machine settings when configuring Elasticsearch, HDFS, and Apache Kafka destinations, or web services to which you send log messages via the HTTP protocol. For details, see:
  - Elasticsearch destination options
  - Elasticsearch2 destination options
  - HDFS destination options
  - HTTP destination options
  - Kafka destination options
  - Global options
- A new HDFS destination option, called `hdfs-append-enabled()` has been added. For further information, see hdfs-append-enabled().
- Macros are now supported in the `hdfs-file()` option. For details, see hdfs-file().
- The following new TLS options have been added:
  - dhparam-file()
  - ecdh-curve-list()
  - pkcs12-file().
- A new parser, capable of processing input in XML format, has been added. For more information, see The XML parser.

Changes in documentation:

- Added section about commercial version of syslog-ng. For more information, see Commercial version of syslog-ng.
- Added warning about the requirement to delete the persist file once the `dir()` option of `disk-buffer()` has been modified or a new one has been added. For more information, see destination: Forward, send, and store log messages.
- Clarified information about the Python parser's `deinit()` method. It runs not only at a syslog-ng graceful stop, but at a reload too. For details, see Methods of the python() parser.
- Several corrections and editorial changes.

Version 3.10 - 3.11

Changes in product:

- Looking up GeoIP2 data from IP addresses has been added to the document.
- http: Posting messages over HTTP without Java has been upgraded with new improvements.
- The geoip() parser is now deprecated. Looking up GeoIP data from IP addresses (DEPRECATED).
- The template() option has been added to the Apache Access Log Parser. For details, see: The Apache Access Log Parser.
- SSL-related options have been added to amqp() destination. For details, see: amqp() destination options.
- The prefix() option has been added to the Cisco parser. For details, see: The Cisco Parser.
- The drop-unmatched() option has been added to the db-parser() statement. For details, see: Using pattern databases.
- The event-time() option has been added to the Riemann destination. For details, see: riemann: Monitoring your data with Riemann.

Changes in documentation:

- A new example has been added to the osquery() source. For details, see: osquery: Collect and parse osquery result logs.
- Several corrections and editorial changes.

Version 3.9 - 3.10

Changes in product:

- wildcard-file: Collecting messages from multiple text files has been added to the document.
- snmptrap: Read Net-SNMP traps has been added to the document.
- osquery: Collect and parse osquery result logs has been added to the document.
- The `elasticsearch2()` destination now supports HTTPS mode, including encryption, and also password- and certificate-based authentication. For details, see elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher.
- The `http()` destination now supports encryption, and also password- and certificate-based authentication. For details, see HTTP destination options.

- The `hdfs()` destination now supports Kerberos authentication. For details, see Kerberos authentication with syslog-ng hdfs() destination.
- The Python Parser has been added to the document.
- The Cisco Parser has been added to the document.
- map-value-pairs: Rename value-pairs to normalize logs has been added to the document.
- The `list-*` template functions allow you to manipulate comma-separated lists. For details, see List manipulation.
- The new `basename()` and `dirname()` template functions allow you to easily separate the path and filenames. For details, see Template functions of syslog-ng OSE.
- stardate has been added to the document.
- create-statement-append() has been added to the document.
- The default value of the `log-msg-size()` option has been increased to 64k. That way syslog-ng OSE will not truncate long log messages, which are getting increasingly common.

Changes in documentation:

- Splunk: Sending log messages to Splunk has been added to the document.
- About disk queue files has been added to the document.
- An example failure script has been added to Running a failure script.
- Several corrections and editorial changes.

Version 3.8 - 3.9

Changes in product:

- When using TLS-transport, you can now use certain fields of the X.509 certificates as macros. For details, see .TLS.X509.
- The `elastic2()` destination driver now supports Search Guard, an alternative security solution for Elasticsearch. For details, see Search Guard and syslog-ng OSE.
- .TLS.X509 has been added to the document.
- Unsetting message fields has been updated with `groupunset()`.

Changes in documentation:

- Corrections and editorial changes.

Version 3.7 - 3.8

Changes in product:

- Enriching log messages with external data has been added to the document.
- Correlating log messages has been added to the document.
- elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher has been added to the document.
- http: Posting messages over HTTP without Java has been added to the document.
- logmatic: Using Logmatic.io has been added to the document.
- loggly: Using Loggly has been added to the document.
- Disk-based buffering has been added to syslog-ng OSE. For details, see Using disk-based and memory buffering.
- What's new in the syslog-ng pattern database format V5, , has been added to Element: create-context has been added to db-parser: Process message content with a pattern database (patterndb).
- Parsing dates and timestamps has been added to parser: Parse and segment structured messages.
- The Apache Access Log Parser has been added to parser: Parse and segment structured messages.
- New options of the `set()` rewrite operator have been added to Setting message fields to specific values.
- A rewrite operator to unset fields has been added to Unsetting message fields.
- A template function that formats name-value pairs as ArcSight Common Event Format extension has been added to format-cef-extension.
- Numerical template functions that work on numerical values of a correlation context have been added to Numerical operations.
- The `inherit-environment()` option has been added to program: Receiving messages from external applications and program: Sending messages to external applications.
- @NLSTRING@ has been added to Using pattern parsers.

Changes in documentation:

- Looking up GeoIP data from IP addresses (DEPRECATED) has been moved to Enriching log messages with external data.
- Several corrections and editorial changes.

Version 3.6 - 3.7

Changes in product:

- mbox: Converting local e-mail messages to log messages has been added to the document.
- The `keep-alive()` option has been added to the `program()` destination.
- The Linux Audit Parser has been added to parser: Parse and segment structured messages.
- python has been added to Template functions of syslog-ng OSE.
- Posting messages over HTTP has been added to the document.
- Write your own custom destination in Java or Python has been added to the document.
- Looking up GeoIP data from IP addresses (DEPRECATED) has been added to the document.
- Elasticsearch destination options has been added to the document.
- kafka: Publishing messages to Apache Kafka has been added to the document.
- hdfs: Storing messages on the Hadoop Distributed File System (HDFS) has been added to the document.
- Parsing key=value pairs has been added to the document.
- format-cim has been added to the document.
- Simple templates can be defined without braces. Templates can also reference other templates. For details, see Templates and macros.
- Custom template functions can be defined in the syslog-ng OSE configuration. For details, see Using template functions.
- CSV-parsers can use strings as delimiters. For details, see delimiters().
- IPv6 addresses can be filtered using a new filter. For details, see netmask6().
- The `loggen` utility can send messages indefinitely using the `--permanent` option.
- The `ssl-options()` option has beed added to TLS options.
- TLS-support has been added to riemann() destination options.
- The `extract-solaris-msgid()` parser has beed added to sun-streams: Collecting messages on Sun Solaris.
- The `context` option of `inherit-properties` has beed added to Actions and message correlation.
- flush-lines() has been added to the document.
- The `sanitize-utf8` flag has been added to the list of source flags.
- The `format-welf` function has been added to Template functions of syslog-ng OSE.

- The `pass-unix-credentials()` option has been added to Global options of syslog-ng OSE.
- The `use-uniqid()` option has been added to Global options of syslog-ng OSE.
- The `UNIQID` macro has been added to Macros of syslog-ng OSE.
- The JSON-parser now handles special characters in object names. For details, see extract-prefix().
- The `syslog-debun` tool used to generate syslog-ng OSE debug bundles has been documented. For details, see The syslog-ng-debun manual page.
- The `--control` option has been added to the The syslog-ng manual page manual page.
- Version 3.7 and newer automatically includes the `plugin.conf` files from the `<directory-where-syslog-ng-is-installed>/scl/*/` directories, making it easier to use and distribute configuration blocks.
- The `--enable-all-modules` compiler option has beed added to Compiling options of syslog-ng OSE.
- The `create-dirs()` option has been added to unix-stream() and unix-dgram() destination options.

Changes in documentation:

- Generating configuration blocks from a script has been added to the document.
- Example: Sending alert when a client disappears has been added to the document.
- The `tcp()`, `tcp6()`, `udp()`, `udp6()` source and destination drivers have been deprecated, as all of their functionality can be achieved with the `network()` driver. For help on migrating to the `network()` driver, see Change an old source driver to the network() driver and Change an old destination driver to the network() driver.
- The beginning of Troubleshooting syslog-ng has been extended with basic troubleshooting information.
- The description of the `chain-hostnames()` global option has been clarified and extended. For details, see chain-hostnames().
- Other editorial corrections.

Version 3.5 - 3.6

Changes in product:

Changes in documentation:

- riemann: Monitoring your data with Riemann has been added to the document.
- nodejs: Receiving JSON messages from nodejs applications has been added to the document.

- systemd-journal: Collecting messages from the systemd-journal system log storage has been added to the document.
- systemd-syslog: Collecting systemd messages using a socket has been added to the document.
- use-rcptid() has been added to the document.
- Setting multiple message fields to specific values has been added to the document.
- The `retries` and `throttle` options are available for the SMTP, MongoDB, AMQP, and Redis destinations.
- The description of the `multi-line-mode` option has been updated.
- UNIX credentials and other metadata has been added to the document.
- RUNID has been added to Macros of syslog-ng OSE.
- The `extract-prefix` option has been added to The JSON parser The JSON parser.
- The `graphite-output`, `or` and `padding` template functions have been added to Template functions of syslog-ng OSE.
- PCRE is now a required dependency of syslog-ng OSE, and by default, syslog-ng OSE uses PCRE-style regular expressions. Therefore, the `--enable-pcre` compliation option has been removed.
- graphite: Sending metrics to Graphite has been added to the document.
- pseudofile() has been added to the document.
- The `custom-domain()` and `stats-lifetime()` options have been added to Global options.
- The `retry_sql_inserts` option has been renamed to `retries` to increase consistency.
- on-error() can be set locally for MongoDB destinations as well. Also, MongoDB destinations support the `username` and `password` options, and connecting to the server using UNIX domain sockets. For details, see mongodb: Storing messages in a MongoDB database.
- How syslog-ng OSE connects the MongoDB server has been added to the document.
- Several typos and syntax errors in examples have been corrected.

Feedback

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

The source of this guide is available on GitHub. In case of the syslog-ng Open Source Edition guides, you can also:

- Open an issue

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.

Introduction to syslog-ng

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

What syslog-ng is

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

Secure and reliable log transfer

The syslog-ng OSE 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 TCP protocol ensures that no messages are lost.

Disk-based message buffering

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

Secure logging using TLS

Log messages may contain sensitive information that should not be accessed by third parties. Therefore, syslog-ng OSE 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 OSE comes with a set of built-in parsers, which you can combine to build very complex things.

Filter and classify

The syslog-ng OSE 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 OSE 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 OSE 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 OSE 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. syslog-ng OSE 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 OSE 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 OSE application is the ideal choice to collect logs in massively heterogeneous environments using several different operating systems and hardware platforms, including Linux, Unix, BSD, Sun Solaris, HP-UX, Tru64, and AIX.

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.

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.

What is new in syslog-ng Open Source Edition 3.17?

Version 3.17 of syslog-ng Open Source Edition includes the following main features.

linux-audit() source driver

A new source driver, `linux-audit()`, has been added. The `linux-audit()` source reads and automatically parses the Linux audit logs. For details, see Administration Guide.

exclude-kmsg() system source option

A new system source option, `exclude-kmsg()` makes it possible to avoid duplicate collection of kernel logs or errors in kernel log collection (for example, in scenarios where the log management on the host system and the containerized solution are collecting the kernel logs simultaneously). When set to `yes`, syslog-ng OSE will omit kernel logs on platforms where they are available separately. For details, see Administration Guide

SCL syntax updates

- You can now refer to any additional parameters at the end of the argument in a block by adding three dots to it (…). It tells syslog-ng OSE that this macro accepts `__VARARGS__`, therefore any name-value pair can be passed without validation. For details, see Administration Guide.
- You can now make parameters mandatory in block definitions by defining them with empty brackets `()`. For details, see Administration Guide.

Enhancements

- The `failover()` option allows you to specify what happens after syslog-ng OSE fails over to a secondary server. Additionally, the `failover-servers()` option has been deprecated and removed from the document. For more information about the `failover()` option, see Administration Guide.
- A note about JVM still running after deleting all Java destinations and reloading

syslog-ng has been added to the description of Java destinations.

- The default value of the `--skip-tokens` parameter of the loggen application has been changed to `0`. For details, see Administration Guide.
- Added support for the timestamp format used by Cisco Unified Call Manager in the Cisco Parser. For details, see the source code of this parser on GitHub.

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

- The source code of syslog-ng Open Source Edition is released under the GPLv2 license and is available on GitHub.
- See the list of precompiled syslog-ng OSE binary packages.

4

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.

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.

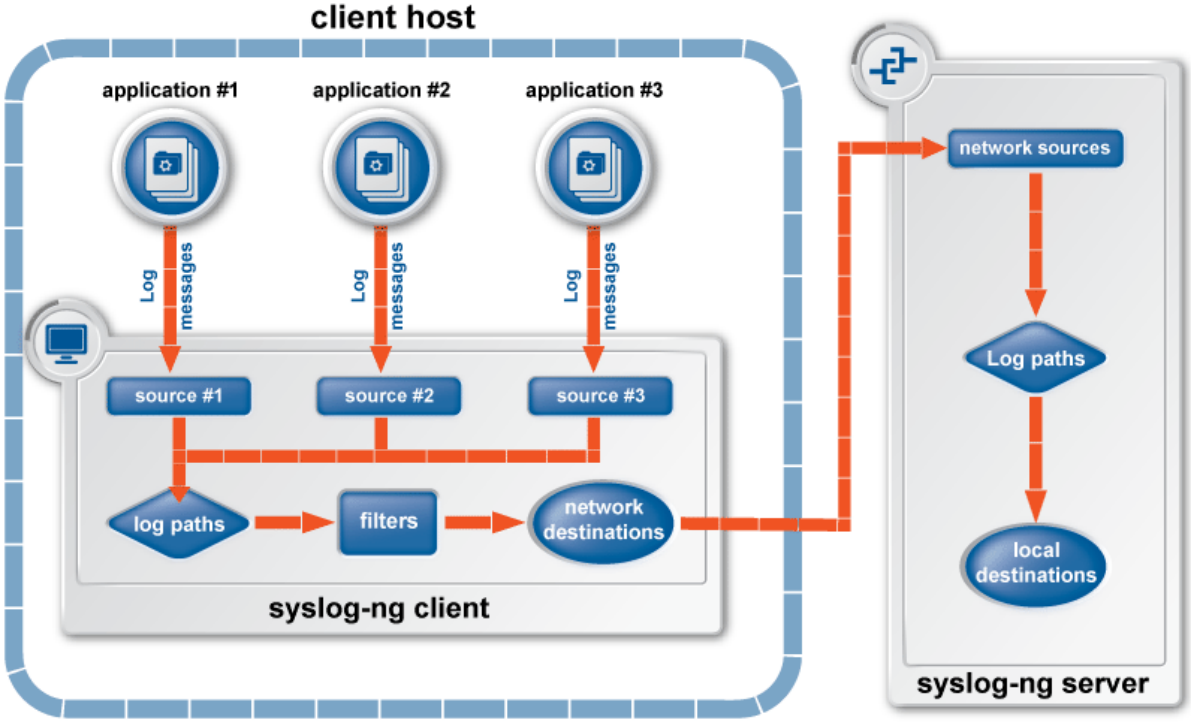
Other optional elements that can appear in log statements are *parsers* and *rewriting rules*. Parsers segment messages into different fields to help processing the messages, while rewrite rules modify the messages by adding, replacing, or removing parts of the messages.

The route of a log message in syslog-ng

Purpose:

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

Figure 1: The route of a log message



Steps:

1. A device or application sends a log message to a source on the syslog-ng client. For example, an Apache web server running on Linux enters a message into the `/var/log/apache` file.
2. The syslog-ng client running on the web server reads the message from its `/var/log/apache` source.
3. The syslog-ng client processes the first log statement that includes the `/var/log/apache` source.
4. The syslog-ng client performs optional operations (message filtering, parsing, and rewriting) on the message, for example, it compares the message to the filters of the log statement (if any). If the message complies with all filter rules, syslog-ng sends the message to the destinations set in the log statement, for example, to the remote syslog-ng server.

CAUTION:

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

NOTE:

The syslog-ng client sends a message to *all* matching destinations by default. As a result, a message may be sent to a destination more than once, if the destination is used in multiple log statements. To prevent such situations, use the `final` flag in the destination statements. For details, see 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:

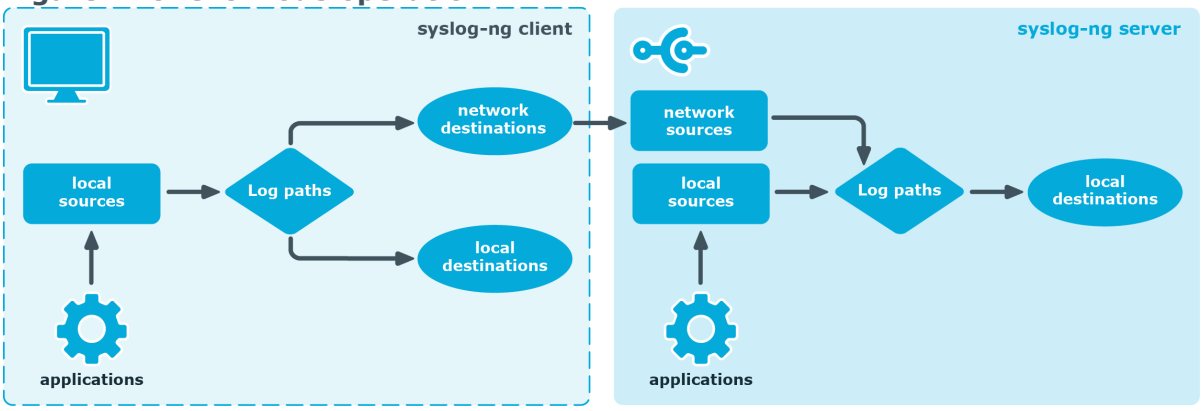
   The syslog-ng application can stop reading messages from its sources if the destinations cannot process the sent messages. This feature is called flow-control and is detailed in Managing incoming and outgoing messages with flow-control.

Modes of operation

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

Client mode

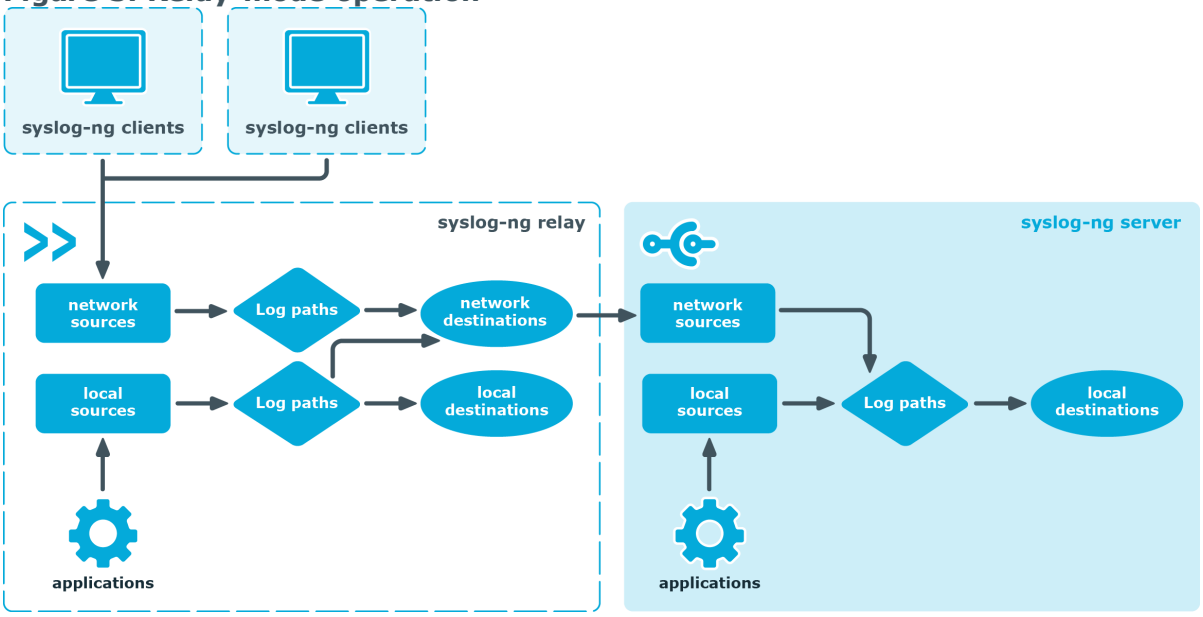
Figure 2: Client-mode operation



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.

Relay mode

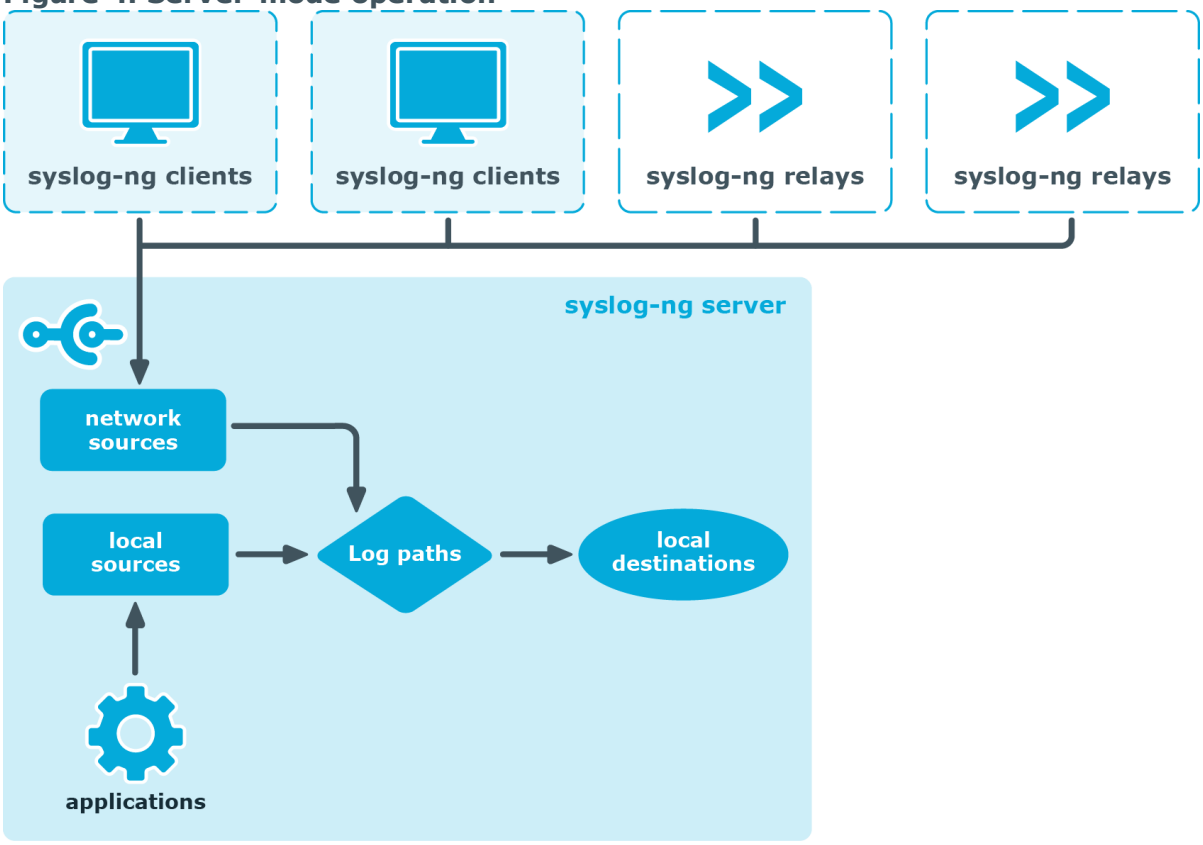
Figure 3: Relay-mode operation



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.

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.

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 source: Read, receive, and collect log messages.
- *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 destination: Forward, send, and store log messages.
- *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 Customize 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 Customize 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 db-parser: Process message content with a pattern database (patterndb).
- *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 log: Filter and route log messages using 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 Customize 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 OSE.

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

Timezones and daylight saving

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

The syslog-ng application supports messages originating from different timezones. The original syslog protocol (RFC3164) does not include timezone information, but syslog-ng provides a solution by extending the syslog protocol to include the timezone in the log messages. The syslog-ng application also enables administrators to supply timezone information for legacy devices which do not support the protocol extension.

How syslog-ng OSE assigns timezone to the message

When syslog-ng OSE 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 OSE 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 OSE application will use the timezone and daylight-saving that was effective when the timestamp was generated. For example, the current time is `2011-03-11` (March 11, 2011) in the `EU/Budapest` timezone. When daylight-saving is active (summertime), the offset is `+02:00`. When daylight-saving is inactive (wintertime) the timezone offset is `+01:00`. If the timestamp of an incoming message is `2011-01-01`, the timezone associated with the message will be `+01:00`, but the timestamp will be converted, because 2011-01-01 meant winter time when daylight saving is not active but the current timezone is `+02:00`.
3. Specify the timezone in the destination driver using the `time-zone()` parameter. Each destination driver might have an associated timezone value: syslog-ng converts message timestamps to this timezone before sending the message to its destination (file or network socket). Each destination defaults to the value of the `send-time-zone()` global option.

   NOTE:

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

   CAUTION:

   If syslog-ng OSE 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 OSE 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 OSE 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.)

Product licensing

Starting with version 3.2, the syslog-ng Open Source Edition application is licensed under a combined LGPL+GPL license. The core of syslog-ng OSE is 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.

NOTE:

Practically, the code stored under the `lib` directory of the source code package is under LGPL, the rest is GPL.

For details about the LGPL and GPL licenses, see GNU Lesser General Public License and GNU General Public License, respectively.

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).

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 OSE: see Message representation in syslog-ng OSE.

BSD-syslog or legacy-syslog messages

This section describes the format of a syslog message, according to the legacy-syslog or BSD-syslog protocol. A syslog message consists of the following parts:

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

The total message cannot be longer than 1024 bytes.

The following is a sample syslog message:

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

The message corresponds to the following format:

```
<priority>timestamp hostname application: message
```

The different parts of the message are explained in the following sections.

NOTE:

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

The PRI message part

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

NOTE:

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

Table 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 the 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
6	Informational: informational messages
7	Debug: debug-level messages

The HEADER message part

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

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

NOTE:

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

The MSG message part

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

IETF-syslog messages

This section describes the format of a syslog message, according to the IETF-syslog protocol. A syslog message consists of the following parts:

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

The following is a sample syslog message (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
```

In this example, the Facility has the value of 4, severity is 2, so PRI is 34. The VERSION is 1. The message was created on 11 October 2003 at 10:14:15pm UTC, 3 milliseconds into the next second. The message originated from a host that identifies itself as "mymachine.example.com". The APP-NAME is "su" and the PROCID is unknown. The

MSGID is "ID47". The MSG is "'su root' failed for lonvick...", encoded in UTF-8. The encoding is defined by the BOM:

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 MSG is "'su root' failed for lonvick...".

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

The PRI message part

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

NOTE:

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

Table 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

Numerical Code	Facility
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 OSE 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 OSE). 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

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

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 follwing 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 OSE has attached to or extracted from the 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 OSE

When the syslog-ng OSE application receives a message, it automatically parses the message. The syslog-ng OSE 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 OSE 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 `${MESSAGE}` 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 OSE. 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 OSE 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 OSE. The syslog-ng OSE 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 OSE 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 OSE, 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 vs. soft macros.

Message size and encoding

Internally, syslog-ng OSE 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 OSE 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 OSE 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 OSE 3.3 and later.

The syslog-ng OSE 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 OSE 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 OSE 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 OSE cannot convert the data into the specified type, an error occurs, and syslog-ng OSE drops the message by default. To change how syslog-ng OSE 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 OSE 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")
)
```

pair()

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

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

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

rekey()

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

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

If you want to modify the names of several message fields, see also map-value-pairs: Rename value-pairs to normalize logs.

- 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`

  This option was called `replace()` in syslog-ng OSE version 3.4.
- `shift("<number>")`
- Cuts the specified number of characters from the beginning of the name.

Example: 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 OSE 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 OSE 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 OSE hosts

When you send your log messages from a syslog-ng OSE client through the network to a syslog-ng OSE 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 OSE you can change many aspects of the network communication. First of all, there is the structure of the messages itself. Currently, syslog-ng OSE 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 OSE. 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. 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(rltp))`
4. `syslog(transport(tls))`
5. `syslog(transport(rltp(tls-required(yes)))`
6. `network(transport(tcp))`
7. `network(transport(udp))`
8. `network(transport(rltp))`
9. `network(transport(tls))`
10. `network(transport(rltp(tls-required(yes)))`
11. `network(transport(tcp) flag(syslog-protocol))`
12. `network(transport(udp) flag(syslog-protocol))`
13. `network(transport(rltp)flag(syslog-protocol))`
14. `network(transport(tls) flag(syslog-protocol))`
15. `network(transport(rltp(tls-required(yes)) flag(syslog-protocol))`

If you use the same driver and options in the destination of your syslog-ng OSE client and the source of your syslog-ng OSE 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 5: 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-/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.

Commercial version of syslog-ng

The syslog-ng application has a commercial version available, called syslog-ng Premium Edition (syslog-ng PE). The commercial version comes with well-tested features from its open source foundation, a number of extra features, enterprise-level support, as well as a

ready-to-use log management appliance built on the strengths of syslog-ng Premium Edition.

Exclusive features related to compliance

Collecting and analyzing log messages is required directly or indirectly by several regulations, frameworks, and standards, including the Sarbanes-Oxley Act (SOX), the Health Insurance and Portability Act (HIPAA), and the Payment Card Industry Data Security Standard (PCI-DSS). syslog-ng PE provides a set of features that help you comply with regulations that require the central collection of log messages in a tamperproof way:

- Logstore files enable you to store log messages securely in encrypted, compressed and timestamped binary files. From a compliance point of view, this serves a double purpose. Encryption guarantees the integrity of log messages so you can be sure that they have not been manipulated. Timestamping provides verifiable proof about the exact time when log messages arrived.
- Reliable Log Transfer Protocol (RLTP) is a proprietary transport protocol that prevents message loss during connection breaks. When using this protocol, the sender detects which messages the receiver has successfully received (based on the acknowledgements returned by the receiver after having processed messages). If messages are lost during 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.

Wide range of supported platforms with binary installers

syslog-ng Premium Edition comes with tested binary files that are available for a wide array of server platforms, reducing the time required for installation and maintenance. Support for a wide range of operating system and hardware platforms also make syslog-ng PE an ideal choice to collect logs in massively heterogeneous environments.

Enterprise-level support services

As all commercial software, syslog-ng PE also comes with various enterprise-level support packages, which means that you get immediate and pro-active assistance (24x7 if you choose a top-tier package), dedicated to resolving your issue as soon as possible when you experience problems.

For more information about syslog-ng Premium Edition, see The syslog-ng Premium Edition Administrator Guide.

syslog-ng Store Box, ready-to-use log management appliance

syslog-ng Store Box (SSB) is a log management appliance that is built on syslog-ng Premium Edition. It is a turnkey solution to manage your log data, meaning that no software installation is necessary. As SSB is available both as a virtual machine and a physical appliance, it is also easily scalable.

SSB provides a number of features that can add value for your use cases:

- A web GUI that makes searching logs, as well as configuring and managing SSB itself easy:
    - The search interface allows you to use wildcards and Boolean operators to perform complex searches, and drill down on the results. You can gain a quick overview and pinpoint problems fast by generating ad-hoc charts from the distribution of the log messages.

      In addition, you can easily create customized reports from the charts and statistics you create on the search interface to demonstrate compliance with standards and regulations such as PCI-DSS, ISO 27001, SOX and HIPAA.
    - Configuring SSB is done through the user interface. All of the flexible filtering, classification and routing features in the syslog-ng Open Source Edition and syslog-ng Premium Edition can be configured with it. Access and authentication policies can be set to integrate with Microsoft Active Directory, LDAP and Radius servers. The web interface is accessible through a network interface dedicated to management traffic. This management interface is also used for backups, sending alerts, and other administrative traffic.
- High availability support to ensure continuous log collection in business-critical environments.

For further details about syslog-ng Store Box, see The syslog-ng Store Box Administrator Guide.

Upgrading from syslog-ng OSE to syslog-ng PE

If you wish to upgrade from syslog-ng OSE to syslog-ng PE, read the blog post Upgrading from syslog-ng OSE to syslog-ng PE for instructions and tips.

Installing syslog-ng

This chapter explains how to install syslog-ng Open Source Edition on various platforms.

- You can install syslog-ng OSE on many platforms using the package manager and official repositories of the platform. For a list of third-party packages available for various Linux, UNIX, and other platforms, see the syslog-ng OSE third-party binaries page.
- For instructions on compiling syslog-ng Open Source Edition from the source code, see Compiling syslog-ng from source.
- You can use a syslog-ng docker image.

  For detailed information on how to run your central log server in Docker and other Docker-related syslog-ng use cases, see the Logging in Docker using syslog-ng white paper.

Compiling syslog-ng from source

Purpose:

To compile syslog-ng Open Source Edition (OSE) from the source code, complete the following steps. Alternatively, you can use precompiled binary packages on several platforms. For a list of third-party packages available for various Linux, UNIX, and other platforms, see the syslog-ng OSE third-party binaries page.

Steps:

1. Download the latest version of syslog-ng OSE from GitHub. The source code is available as a tar.gz archive file.
2. Install the following packages that are required to compile syslog-ng. These packages are available for most UNIX/Linux systems. Alternatively, you can also download the sources and compile them.
   - A version of the *gcc* C compiler that properly supports Thread Local Storage (TLS), for example, version 4.5.
   - The *GNU flex* lexical analyser generator, available here.

- The *bison* parser generator, available here.
- The development files of the *glib* library, available here.
- The development files of the *Autoconf Archive* package, available here.
- The syslog-ng OSE application now uses PCRE-type regular expressions by default. It requires the `libpcre` library package, available here.
- If you want to use the Java-based modules of syslog-ng OSE (for example, the Elasticsearch, HDFS, or Kafka destinations), you must compile syslog-ng OSE with Java support.
    - Download and install the Java Runtime Environment (JRE), 1.7 (or newer). You can use OpenJDK or Oracle JDK, other implementations are not tested.
    - Install gradle version 2.2.1 or newer.
    - Set `LD_LIBRARY_PATH` to include the `libjvm.so` file, for example:`LD_LIBRARY_PATH=/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/amd64/server:$LD_LIBRARY_PATH`

      Note that many platforms have a simplified links for Java libraries. Use the simplified path if available. If you use a startup script to start syslog-ng OSE set `LD_LIBRARY_PATH` in the script as well.
    - If you are behind an HTTP proxy, create a `gradle.properties` under the `modules/java-modules/` directory. Set the proxy parameters in the file. For details, see The Gradle User Guide.

3. If you want to post log messages as HTTP requests using the `http()` destination, install the development files of the *libcurl* library. This library is not needed if you use the `--disable-http` compile option. Alternatively, you can use a Java-based implementation of the HTTP destination.
4. If you want to use the spoof-source function of syslog-ng, install the development files of the *libnet* library, available here.
5. If you want to send e-mails using the `smtp()` destination, install the development files of the *libesmtp* library. This library is not needed if you use the `--disable-smtp` compile option.
6. If you want to use the */etc/hosts.deny* and */etc/hosts.allow* for TCP access, install the development files of the *libwrap* (also called TCP-wrappers) library, available here.
7. Enter the new directory and issue the following commands. (If the `./configure` file does not exist, for example, because you cloned the repository from GitHub instead of using a release tarball, execute the `./autogen.sh` command.)

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

8. Uncompress the syslog-ng archive using the

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

   or the

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

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

9. Enter the new directory and issue the following commands:

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

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

> NOTE:
>
> - On Solaris, use `gmake` (GNU make) instead of `make`.
> - To build syslog-ng OSE with less verbose output, use the `make V=0` command. This results in shorter, less verbose output, making warnings and other anomalies easier to notice. Note that silent-rules support is only available in recent automake versions.

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

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

> NOTE:
>
> You can also use *--disable options*, to explicitly disable a feature and override autodetection. For example, to disable the TCP-wrapper support, use the *--disable-tcp-wrapper* option. For the list of available compiling options, see Compiling options of syslog-ng OSE.

> **CAUTION:**
>
> **The default linking mode of syslog-ng is `dynamic`. This means that syslog-ng might not be able to start up if the `/usr` directory is on NFS. On platforms where syslog-ng is used as a system logger, the `--enable-mixed-linking` is preferred.**

Compiling options of syslog-ng OSE

When compiling syslog-ng OSE from source, you can use the following compiling options.

- *--enable-all-modules* This option will turn on or off all modules and most features when enabled, unless a feature is explicitly disabled, or not detected automatically. Currently, this means that you must explicitly enable the `pacct()` source, since it is not detected automatically (all other modules are compiled automatically if the required libraries are available).

This also means that the Sun Streams source is enabled on every platform, not only on Solaris, causing a compile error. Use `--enable-all-modules` together with `--disable-sun-streams`.

- *--disable-http* Disable support for the `http()` destination that is based on *libcurl*.
- *--disable-python* Disable support for Python-based modules.
- *--disable-json* Disable JSON support. It also disables `json-parser`, and the `format-cim` and `format-json` template functions. Also, it disables JSON support even if the `json-c` library is installed and detected (see *--enable-json*).
- *--disable-smtp* Disable SMTP support. By default, SMTP support is enabled if the libesmtp library is detected.
- *--enable-amqp* Enable the amqp destination (enabled by default). The source of the RabbitMQ client is included in the source code package of syslog-ng OSE. To use an external client instead, use the `--with-librabbitmq-client=system` compiling option. For details on using this destination, see amqp: Publishing messages using AMQP.
- *--enable-debug* Include debug information.
- *--enable-dynamic-linking* Compile syslog-ng as a completely dynamic binary. If not specified syslog-ng uses mixed linking (`--enable-mixed-linking`): it links dynamically to system libraries and statically to everything else.
- *--enable-geoip* Enable GEOIP support, required for the `geoip2` template function and the `geoip2-parser` (enabled automatically if the `libmaxminddb` library is detected).
- *--enable-ipv6* Enable IPv6 support.
- *--enable-java* Enable support for Java-based modules. For other requirements, see the description of the Java-based module (for example, Prerequisites) that you want to use.
- *--enable-java-modules* Compile the Gradle projects of every Java module available in `modules/java-modules`.
- *--enable-json* Enables JSON support (by default, it uses the `json-c` library included in the source code package of syslog-ng OSE). JSON support is required for `json-parser`, and the `format-cim` and `format-json` template functions.
- *--enable-linux-caps* Enable support for capabilities on Linux. For details, see The syslog-ng manual page.
- *--enable-mongodb* Enable the mongodb destination (enabled by default). The source of the MongoDB client is included in the source code package of syslog-ng OSE. To use an external MongoDB client instead, use the `--with-libmongo-client=system` compiling option. For details on using this destination, see mongodb: Storing messages in a MongoDB database.
- *--enable-pacct* Enable using the `pacct()` driver to collect process-accounting logs on Linux systems.
- *--enable-python* Enable support for Python-based modules.
- *--enable-redis* Enable the redis destination (enabled by default). The source of the libhiredis client (0.11 or newer) must be available. To specify the location of the

library, use the `--with-libhiredis=<path-to-libhiredis>` compiling option. For details on using this destination, see redis: Storing name-value pairs in Redis.

- *--enable-riemann* Enable the riemann destination (enabled by default). The source of the libriemann client must be available. For details on using this destination, see riemann: Monitoring your data with Riemann.
- *--enable-spoof-source* Enable spoof_source feature (disabled by default).
- *--enable-sql* Enables the sql() destination (enabled automatically if the `libdbi` library version 0.9 or newer is installed and detected).
- *--enable-ssl* Enable SSL support, required for encrypted message transfer, as well as template functions that calculate hashes and UUIDs (enabled automatically if the `libopenssl` library is detected).
- *--enable-sun-door* Enable Sun door support even if not detected (autodetected by default).
- *--enable-sun-streams* Enable Sun STREAMS support even if not detected (autodetected by default).
- *--enable-systemd* Enable systemd support on Linux platforms (autodetected by default) (enabled automatically if the `libsystemd-daemon` library is detected).
- *--enable-tcp-wrapper* Enable using */etc/hosts.deny* and */etc/hosts.allow* for TCP access (enabled automatically if the `libwrap` libraries are detected).
- *--with-embedded-crypto* If this option is set, the crypto library is linked directly into libsyslog-ng: the sources of libsyslog-ng-crypto will be appended to the libsyslog-ng sources, and -crypto is not built.
- *--with-ivykis* Specifies which ivykis implementation to use (default value: internal). The source of ivykis is included in the source code package of syslog-ng OSE and is used by default. To use an external implementation instead, use the `--with-ivykis=system` compiling option.
- *--with-libcurl* Specifies the path to the libcurl library. For details on using this destination, see http: Posting messages over HTTP without Java.
- *--with-libhiredis* Specifies the path to the libhiredis library (0.11 or newer). For details on using this destination, see redis: Storing name-value pairs in Redis.
- *--with-libmongo-client* Specifies which MongoDB client to use (default value: internal). The source of the mongodb client is included in the source code package of syslog-ng OSE. To use an external MongoDB client instead, use the `--with-libmongo-client=system` compiling option. For details on using this destination, see mongodb: Storing messages in a MongoDB database.
- *--with-librabbitmq-client* Specifies which RabbitMQ client to use (default value: internal). The source of the rabbitmq client is included in the source code package of syslog-ng OSE and is used by default. To use an external client instead, use the `--with-librabbitmq-client=system` compiling option. For details on using this destination, see amqp: Publishing messages using AMQP.
- *--with-module-dir* Specifies a single directory where the syslog-ng OSE Makefile will install the modules.

- *--module-install-dir* Specifies syslog-ng OSE's module installation directory (normally `$prefix/lib/syslog-ng`). All Java-based SCLs use this option.
- *--with-module-path* Specifies a colon-separated (:) list of directories, where the syslog-ng OSE binary will search for modules.
- *--with-python* Specifies which Python version to use, for example, `--with-python=2.7`
- *--with-timezone-dir* Specifies the directory where syslog-ng looks for the timezone files to resolve the `time-zone()` and `local-time-zone()` options. If not specified, the `/opt/syslog-ng/share/zoneinfo/` and `/usr/share/zoneinfo/` directories are checked, respectively. Note that HP-UX uses a unique file format (`tztab`) to describe the timezone information, but that format is currently not supported in syslog-ng. As a workaround, copy the zoneinfo files from another, non-HP-UX system to the `/opt/syslog-ng/share/zoneinfo/` directory of your HP-UX system.
- *--without-compile-date* Removes the compilation date from the binary. For example, as openSUSE checks if recompilation changes the binary to detect if dependent packages need to be rebuilt or not, and including the date changes the binary every time.

Uninstalling syslog-ng OSE

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

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

Configuring Microsoft SQL Server to accept logs from syslog-ng

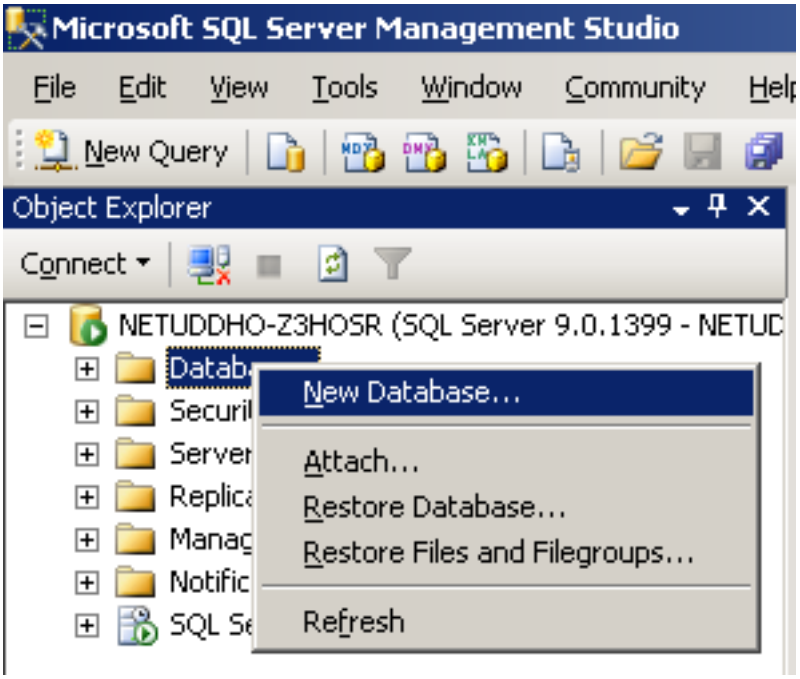
Purpose:

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

Steps:

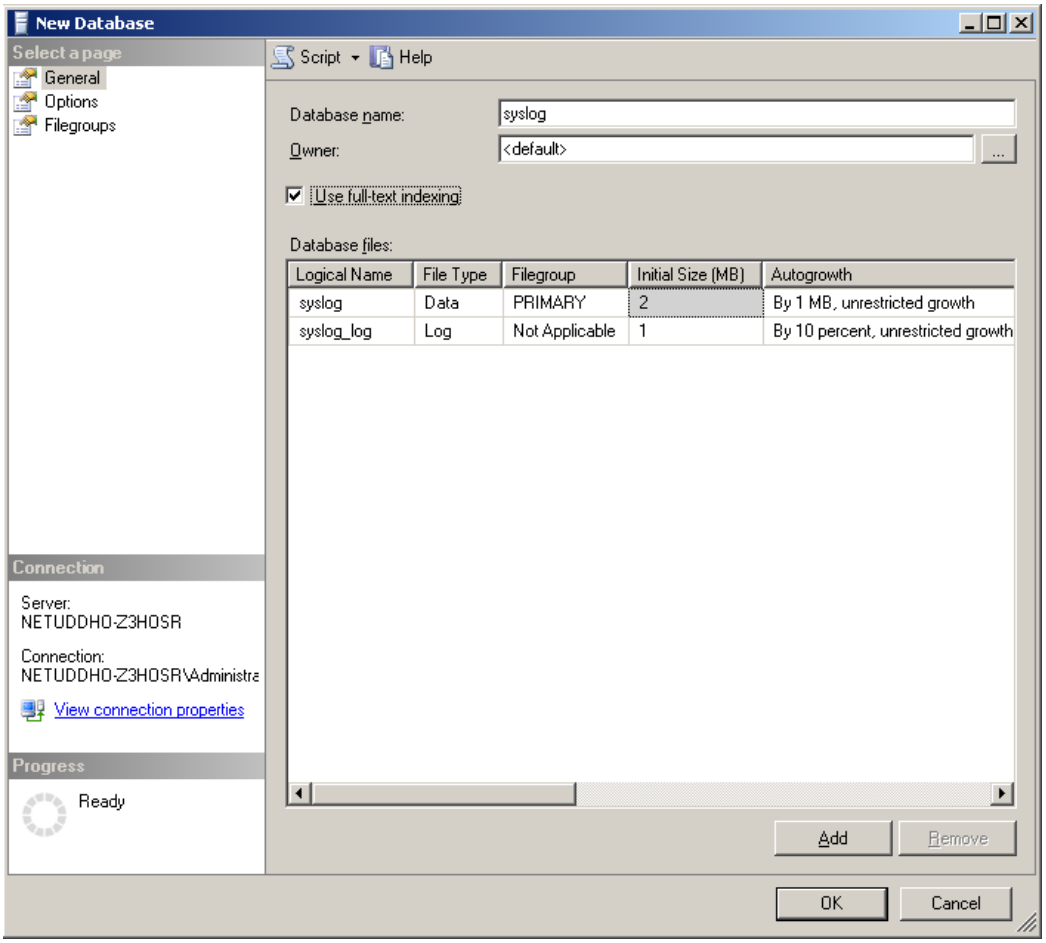
1. Start the SQL Server Management Studio application. Select **Start > Programs > Microsoft SQL Server 2005 > SQL Server Management Studio**.
2. Create a new database.

    a. **Figure 5: Creating a new MSSQL database 1.**



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

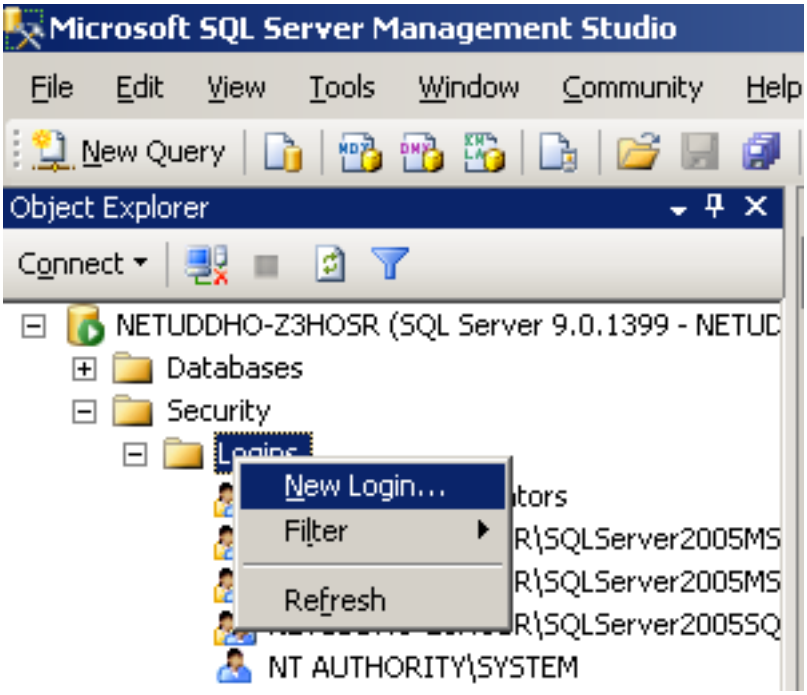
b. **Figure 6: Creating a new MSSQL database 2.**



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

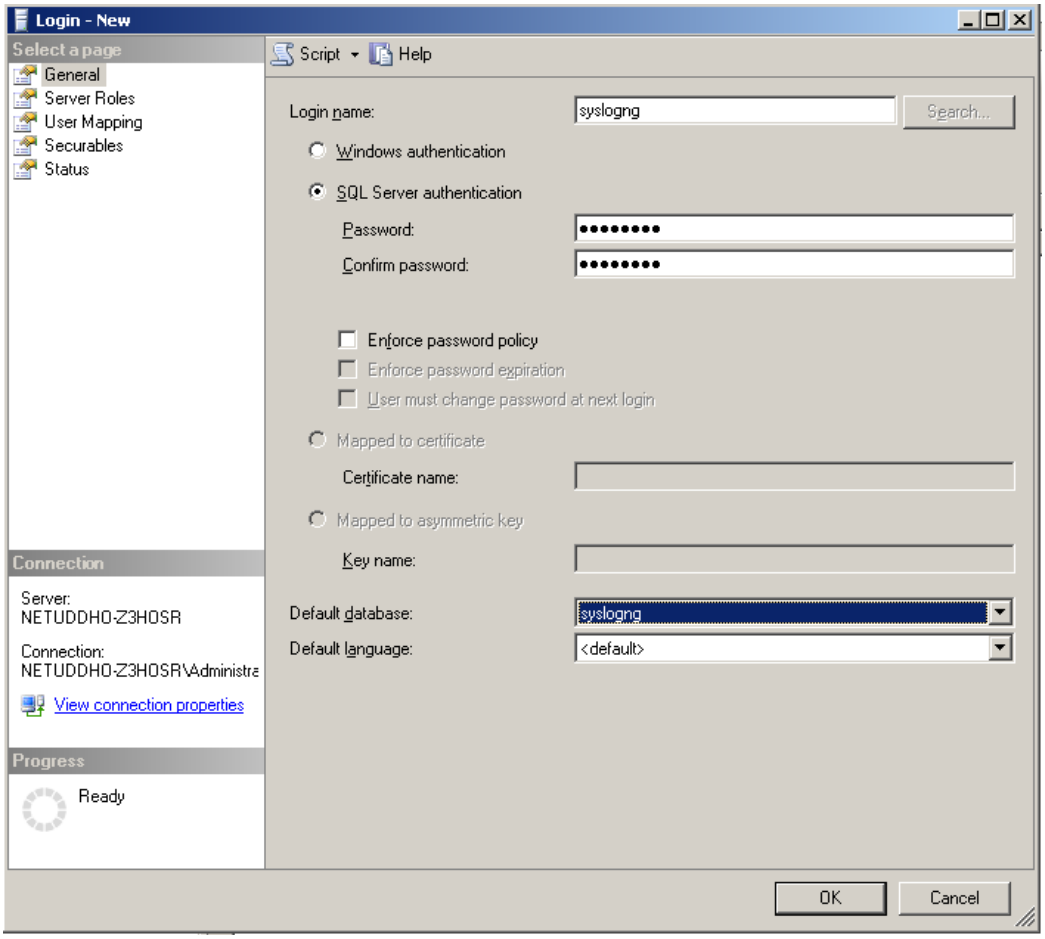
3. Create a new database user and associate it with the new database.

a. **Figure 7: Creating a new MSSQL user 1.**



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

b. **Figure 8: 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.

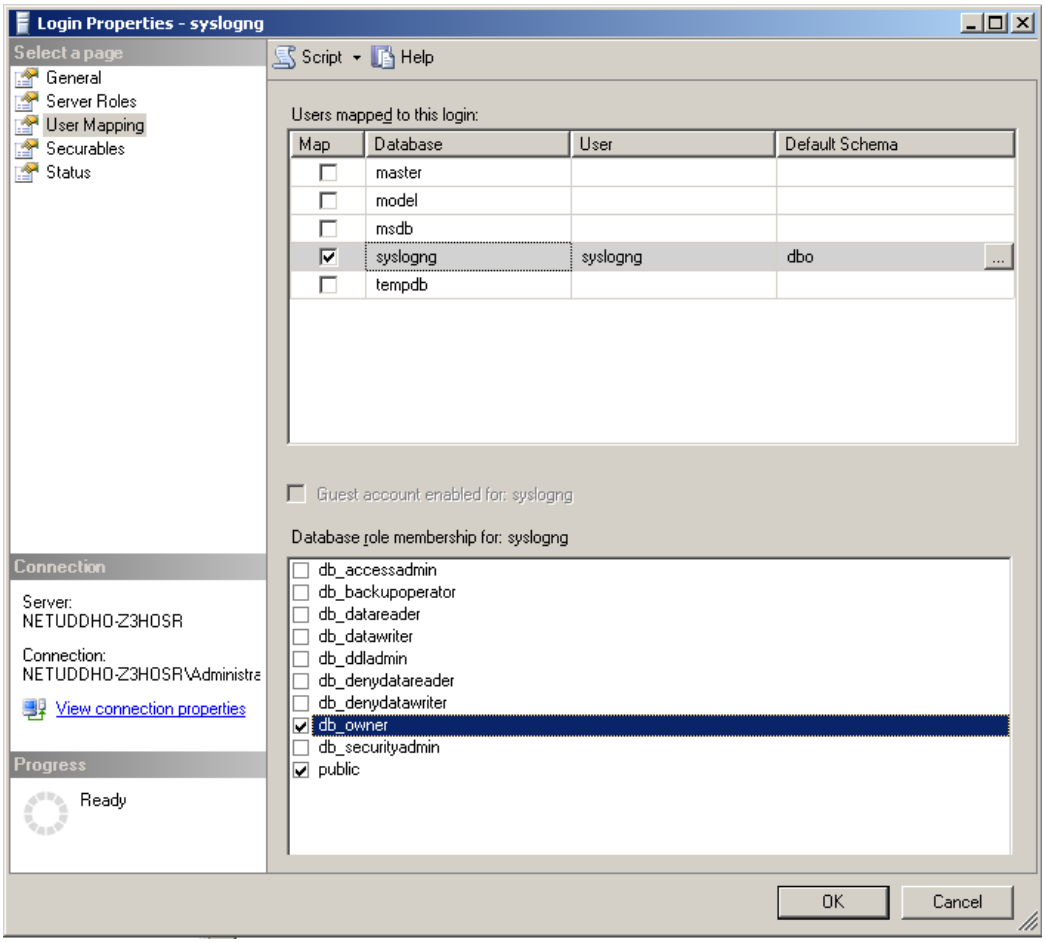
d. In the **Default database** field, select the database created in Step 2 (for example `syslogng`).

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.**

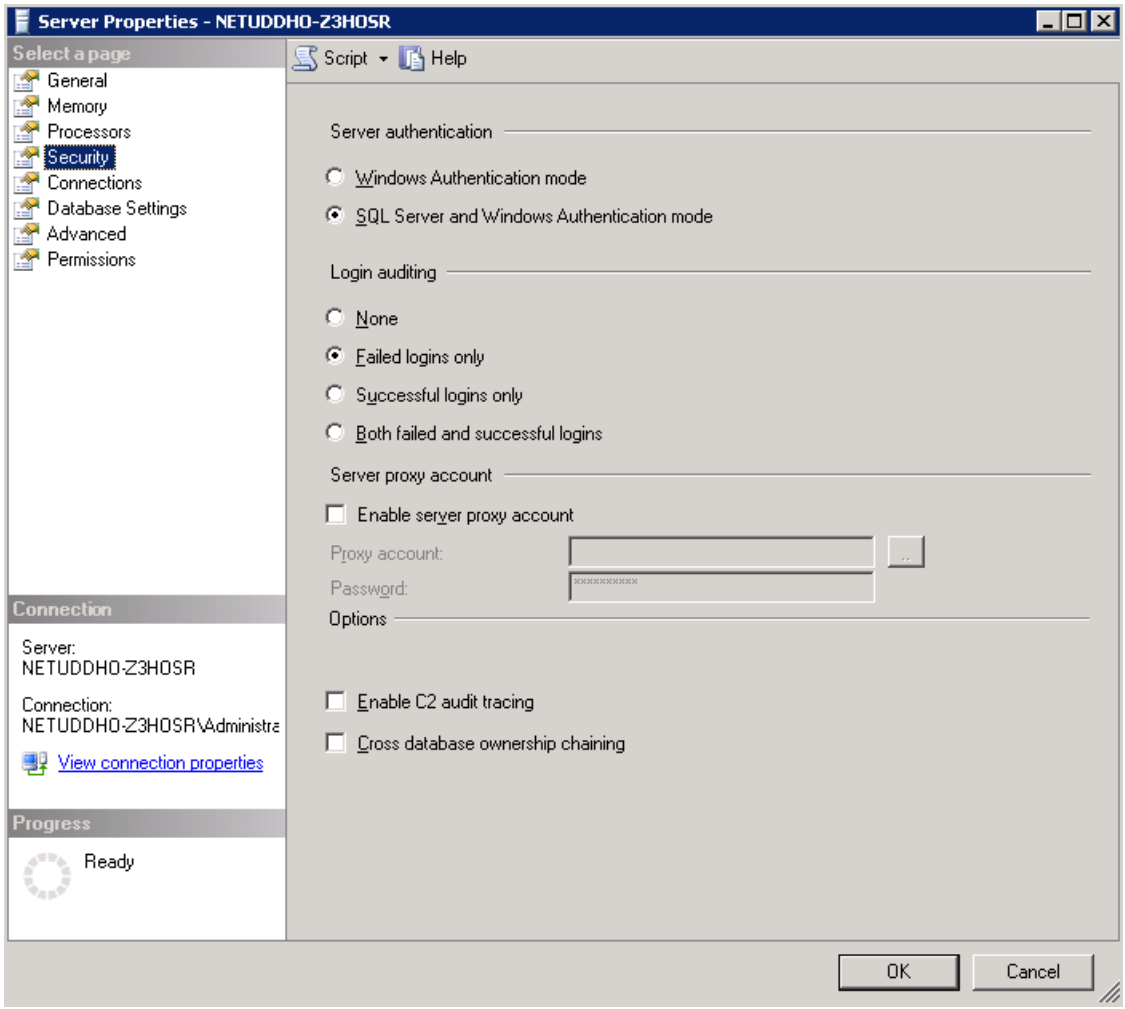
f. In the Object Explorer, select **Security > Logins**, then right-click on the new login created in the previous step, and select **Properties**.

g. **Figure 9: 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 10: 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 OSE quick-start guide

This chapter provides a very brief introduction into configuring the syslog-ng OSE 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 OSE as a client that sends log messages to a central logserver, see Configuring syslog-ng on client hosts.
- To configure syslog-ng OSE as a server that receives log messages from client hosts, see Configuring syslog-ng on server hosts.
- To configure syslog-ng OSE as a relay that receives log messages from client hosts and forwards them to a central logserver, see Configuring syslog-ng on server hosts.

Configuring syslog-ng on client hosts

Purpose:

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

Steps:

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

   Configure the local sources to collect the log messages of the host. Starting with version 3.2, syslog-ng OSE 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.

   To configure syslog-ng OSE, edit the `syslog-ng.conf` file with any regular text editor application. The location of the configuration file depends on how you installed syslog-ng OSE. Native packages of a platform (like the ones downloaded from Linux

repositories) typically place the configuration file under the `/etc/syslog-ng/` directory.

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 OSE collects platform-specific log messages and the internal log messages of syslog-ng OSE.
>
> ```
> source s_local {
>     system();
>     internal();
> };
> ```

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

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

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

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

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

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

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

NOTE:

The default configuration of syslog-ng OSE 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 OSE 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 OSE

The following is the default configuration file of syslog-ng OSE 3.17. It collects local log messages and the log messages of syslog-ng OSE and saves them in the /var/log/messages file.

```
@version: 3.17
@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: 3.17
@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);
};
```

Configuring syslog-ng on server hosts

Purpose:

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

Steps:

1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see Installing syslog-ng.
2. Starting with version 3.2, syslog-ng OSE 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. To configure syslog-ng OSE, edit the `syslog-ng.conf` file with any regular text editor application. The location of the configuration file depends on how you installed syslog-ng OSE. Native packages of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` directory.

   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— OBSOLETE.

> NOTE:
>
> Starting with syslog-ng OSE 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 OSE 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 template and rewrite: Format, modify, and manipulate log 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 Open Source Edition that collects incoming log messages and stores them in a text file.

```
@version: 3.17
@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);
};
```

Configuring syslog-ng relays

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

Configuring syslog-ng on relay hosts

Purpose:

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

Steps:

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

   NOTE:

   It is recommended to use these options on your syslog-ng OSE 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: 3.17
@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 OSE 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 OSE 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 OSE 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 OSE rewriting the HOST field to the address of the host that sent the message to `syslog-ng-server`, which is `syslog-ng-relay` in this case.
- If you cannot or do not want to resolve the `192.168.1.2` IP address on `syslog-ng-relay`, but want to store your log messages on `syslog-ng-server` using the IP address of the original host (that is, `client-host`), you can enable the `spoof-source()` option

on `syslog-ng-relay`. However, `spoof-source()` works only under the following conditions:

- The syslog-ng OSE binary has been compiled with the `--enable-spoof-source` option.
- The log messages are sent using the highly unreliable UDP transport protocol. (Extremely unrecommended.)

The syslog-ng OSE configuration file

Location of the syslog-ng configuration file
The configuration syntax in detail
Notes about the configuration syntax
Defining configuration objects inline
Using channels in configuration objects
Global and environmental variables
Modules in syslog-ng OSE
Managing complex syslog-ng configurations

Location of the syslog-ng configuration file

To configure syslog-ng OSE, edit the `syslog-ng.conf` file with any regular text editor application. The location of the configuration file depends on how you installed syslog-ng OSE. Native packages of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` 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 3.17, this line looks like:

```
@version: 3.17
```

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: 3.17
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 OSE should use DNS resolution to resolve IP addresses. Global options are detailed in Global options of syslog-ng OSE.

  ```
  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 OSE version 3.5, 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 OSE 3.4, you can define configuration objects inline, where they are actually used, without having to define them in a separate placement. This is useful if you need an object only once, for example, a filter or a rewrite rule. 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 OSE 3.4, 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 OSE version 3.2, 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.

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 OSE 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));
};
```

Modules in syslog-ng OSE

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

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

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

For details on the command-line parameters of syslog-ng OSE mentioned in the previous list, see the syslog-ng OSE man page at The syslog-ng manual page.

Loading modules

The syslog-ng Open Source 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 OSE 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.

Use the @requires statement to ensure that the specified module is loaded

To ensure that a module is loaded, include the following statement in the syslog-ng OSE configuration file or the external files included in the configuration file:

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

NOTE:

If you include the `@requires` statement in the:

- syslog-ng OSE configuration file, syslog-ng OSE attempts to load the required module. If it fails to load the module, syslog-ng OSE stops and an error message is displayed.
- external files included in the configuration file, syslog-ng OSE attempts to load the required module. If it fails to load the module, only the external file is not processed.

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 "<filename>"
```

This imports the entire file into the configuration of syslog-ng OSE, 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 OSE will include every matching file. For details on using wildcard characters, see glob.
- A directory. When including a directory, syslog-ng OSE 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:

- 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: 3.17
  @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 3.7 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 OSE 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: 3.17
@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 OSE 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 OSE configuration when syslog-ng OSE is started, the block can be generated dynamically using an external script if needed. This is useful when you are running syslog-ng OSE 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 OSE 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 OSE 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

Purpose:

The syslog-ng OSE application can automatically execute scripts when it is started, and can include the output of such script in the configuration file. To create and use a script that generates a part of the syslog-ng OSE configuration file (actually, a configuration block), complete the following steps. 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 OSE configuration snippet.

Steps:

1. Navigate to the directory where you have installed syslog-ng OSE (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 OSE 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 &abbrev; 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 OSE 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 3.7 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 OSE 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 OSE 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.

source: Read, receive, and collect log messages

How sources work
default-network-drivers: Receive and parse common syslog messages
internal: Collecting internal messages
file: Collecting messages from text files
wildcard-file: Collecting messages from multiple text files
linux-audit: Collecting messages from Linux audit logs
network: Collecting messages using the RFC3164 protocol (network() driver)
nodejs: Receiving JSON messages from nodejs applications
mbox: Converting local e-mail messages to log messages
osquery: Collect and parse osquery result logs
pipe: Collecting messages from named pipes
pacct: Collecting process accounting logs on Linux
program: Receiving messages from external applications
snmptrap: Read Net-SNMP traps
sun-streams: Collecting messages on Sun Solaris
syslog: Collecting messages using the IETF syslog protocol (syslog() driver)
system: Collecting the system-specific log messages of a platform
systemd-journal: Collecting messages from the systemd-journal system log storage
systemd-syslog: Collecting systemd messages using a socket
tcp, tcp6, udp, udp6: Collecting messages from remote hosts using the BSD syslog protocol— OBSOLETE
unix-stream, unix-dgram: Collecting messages from UNIX domain sockets
stdin: Collecting messages from the standard input stream

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 OSE 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 log: Filter and route log messages using 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 6: 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)};`.
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 7: 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.
nodejs()	Receives JSON messages from nodejs applications.
mbox()	Read e-mail messages from local mbox files, and convert them to multiline log messages.
osquery()	Run osquery queries, and convert their results into log messages.
pacct()	Reads messages from the process accounting logs on Linux.
pipe()	Opens the specified named pipe and reads messages.
program()	Opens the specified application and reads messages from its standard output.
snmptrap()	Read and parse the SNMP traps of the Net-SNMP's snmptrapd application.
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 OSE 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.

Name	Description
unix-stream()	Opens the specified unix socket in SOCK_STREAM mode and listens for incoming messages.
stdin()	Collects messages from the standard input stream.
wildcard-file()	Reads messages from multiple files and directories.

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 3.16 and later.

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

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

Also, make sure that your SELinux, AppArmor, and firewall settings permit syslog-ng Open Source 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. Currently it uses the following procedures.

CAUTION:

If you do not configure the TLS keys to dislay to the clients, syslog-ng OSE 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 OSE 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 OSE 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 OSE (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 601 and 6514), syslog-ng OSE 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 OSE (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 3.16 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

Uses the value of the global option if not specified.

rfc5424-tcp-port()

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

Type:	number
Default:	601

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 OSE 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. 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 OSE 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.

normalize-hostnames()

Accepted values:	yes | no
Default:	no

Description: If enabled (`normalize-hostnames(yes)`), syslog-ng OSE 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.

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 character set (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This

is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.

- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng OSE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng OSE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${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 3.16 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.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string

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

Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE).

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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 OSE 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 OSE 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 OSE 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 OSE application provides the following methods to process multi-line messages:

- 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 OSE 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 OSE 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 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.

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 OSE version 3.10 and later.

The syslog-ng OSE 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 OSE is restarted, it records the position of the last sent log message in the persist file, and continues to send messages from this position after the restart. The location of the persist file depends on the package you installed syslog-ng OSE from, typically it is /var/lib/syslog-ng/syslog-ng.persist or /opt/syslog-ng/var/syslog-ng.persist.

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).
- 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 OSE 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 OSE 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 OSE actually monitor. The default value of max-files() is 100.
- If the message does not have a proper syslog header, syslog-ng OSE treats messages received from files as sent by the user facility. Use the default-facility() and default-priority() options in the source definition to assign a different facility if needed.
- For every message that syslog-ng OSE reads from the source files, the path and name of the file is available in the ${FILE_NAME} macro.

Required parameters: base-dir(), filename-pattern(). For the list of available optional parameters, see wildcard-file() source options on page 125.

Example: Using the wildcard-file() driver

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

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

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() on page 137.

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 character set (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng OSE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng OSE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${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 3.16 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

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

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.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE).

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
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 OSE 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 OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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 OSE 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 OSE 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 OSE 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 OSE uses it automatically to detect changes to the source files. If inotify is not available, syslog-ng OSE polls the files as set in the follow-freq() option. To force syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE application provides the following methods to process multi-line messages:

- 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 OSE 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 OSE 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 OSE monitors every subdirectory of the path set in the base-dir() on page 125 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 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.

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

It 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 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 OSE.

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 `file()` driver has the following options:

filename()

Type:	path
Default:	

Description: The log file of `linux-audit`. The syslog-ng OSE 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 OSE. Note that if you use an empty prefix (`prefix("")`) or one starting with a dot, syslog-ng OSE

might replace the original value of an existing macro (note that only soft macros can be overwritten, see Hard vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

`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.

- 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/601 port. To limit accepted connections to only one interface, use the `localip()` parameter. To listen on IPv6 addresses, use the `ip-protocol(6)` option.

Example: Using the network() driver

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

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

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

```
source s_network {
    network(
        ip("192.168.1.1")
```

```
        transport("udp")
    );
};
```

TCP source listening on the IPv6 localhost, port 2222:

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

A TCP source listening on a TLS-encrypted channel.

```
source  s_network  {
       network(
              transport("tcp")
              port(2222)
              tls(peer-verify("required-trusted")
                     key-file("/opt/syslog-ng/etc/syslog-ng/syslog-ng.key")
                     cert-file("/opt/syslog-ng/etc/syslog-ng/syslog-ng.crt")
              );
       );
 };
```

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

network() source options

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

encoding()

Type:	string
Default:	

Description: Specifies the character set (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 3.16 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 OSE.

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

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.

---

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

NOTE: The syslog-ng OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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-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 OSE 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 OSE 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-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 OSE 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 OSE 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 OSE. For details on using name resolution in syslog-ng OSE, see Using name resolution in syslog-ng.
- s

NOTE:

If the log message does not contain a hostname in its `HOST` field, syslog-ng OSE 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 OSE 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 OSE server and also on every relay, otherwise syslog-ng OSE 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 OSE).

listen-backlog()

Type:	integer
Default:	256

Description: Available only for stream based transports (`unix-stream`, `tcp`, `tls`). In TCP, connections are treated incomplete until the three-way handshake is completed between the server and the client. Incomplete connection requests wait on the TCP port for the listener to accept the request. The `listen-backlog()` option sets the maximum number of incomplete connection requests. For example:

```
source  s_network  {
       network(
             ip("192.168.1.1")
             transport("tcp")
             listen-backlog(2048)
             );
};
```

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections, otherwise `log-iw-size()` is divided by 10 (the default value of the `max-connections()` option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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: 601 In case of UDP transport: 514

Description: The port number to bind to.

program-override()

Type:	string
Default:	

Description: Replaces the ${PROGRAM} part of the message with the parameter string. For example, to mark every message coming from the kernel, include the `program-override ("kernel")` option in the source containing `/proc/kmsg`.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make syslog-ng send messages to a broadcast address. For details, see the `socket(7)` manual page.

so-keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. For details, see the `socket(7)` manual page.

so-rcvbuf()

Type:	number
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 OSE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

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

so-sndbuf()

Type:	number
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 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.

transport()

Type:	udp, tcp, or tls
Default:	tcp

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

CAUTION:

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

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

tls()

Type:	tls options
Default:	n/a

Description: This option sets various options related to TLS encryption, for example, key/certificate files and trusted CA locations. TLS can be used only with tcp-based transport protocols. For details, see 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 OSE application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all

hosts which may get to syslog-ng are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

`nodejs`: Receiving JSON messages from nodejs applications

Using the `nodejs()` driver, syslog-ng OSE can receive application logs directly from nodejs applications that use the widespread Winston logging API. The syslog-ng OSE application automatically adds the `.nodejs.winston.` prefix to the name of the fields the extracted from the message.

To use the `nodejs()` driver, the `scl.conf` file must be included in your syslog-ng OSE configuration:

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

The `nodejs()` driver is actually a reusable configuration snippet configured to receive log messages using the `network()` 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 the nodejs configuration snippet on GitHub.

Example: Using the nodejs() driver

The following example uses the default settings of the driver, listening for messages on port 9003 of every IP address of the syslog-ng OSE host.

```
@include "scl.conf"
source apps { nodejs(); };
```

The following example listens only on IP address 192.168.1.1, port 9999.

```
@include  "scl.conf"
 source  apps  {
       nodejs(
             localip(192.168.1.1)
             port(9999)
       )
 };
```

NOTE:

For details on the parameters of the nodejs() driver, see nodejs() source options.

nodejs() source options

The nodejs() driver has the following options.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()

Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

ip() or localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to. By default, syslog-ng OSE 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 OSE automatically joins the necessary multicast group. TCP does not support multicasting.

port() or localport()

Type:	number
Default:	9003

Description: The port number to bind to.

`mbox`: Converting local e-mail messages to log messages

Using the `mbox()` driver, syslog-ng OSE can read e-mail messages from local mbox files, and convert them to multiline log messages.

This driver has only one required option, the filename of the mbox file. To use the `mbox()` driver, the `scl.conf` file must be included in your syslog-ng OSE configuration:

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

The `mbox()` driver is actually a reusable configuration snippet configured to read log messages using the `file()` driver. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of the configuration snippet on GitHub.

Example: Using the mbox() driver

The following example reads the e-mails of the root user on the syslog-ng OSE host.

```
@include  "scl.conf"
 source  root-mbox  {
       mbox("/var/spool/mail/root");
 };
```

mbox() source options

The `mbox()` driver has the following option.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

`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 OSE 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 OSE 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 field, and so on).
- Send the messages to a central location, for example, to Elasticsearch, directly from syslog-ng OSE.

The syslog-ng OSE 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 OSE version 3.10 and later.

Prerequisites:

- To use the `osquery()` driver, the `scl.conf` file must be included in your syslog-ng OSE configuration:

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

- syslog-ng OSE 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 OSE 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:  3.10
@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: 3.10
@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 OSE 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 OSE configuration:

```
@version: 3.10
@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 logs directly to Elasticsearch and Kibana 2.0 or higher. For other destinations, see destination: Forward, send, and store log messages.

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 OSE application reads the messages from this file.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE. 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 vs. 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 OSE uses the flags(no-hostname) option for pipes, meaning that syslog-ng OSE 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, kernel, no-hostname, no-multi-

	line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 3.16 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.

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 OSE 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 OSE to execute external applications.

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

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE).

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections, otherwise `log-iw-size()` is divided by 10 (the default value of the `max-connections()` option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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 OSE 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 OSE 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 OSE 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 OSE application provides the following methods to process multi-line messages:

- 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 OSE 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 OSE 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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 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.

pacct: Collecting process accounting logs on Linux

Starting with version 3.2, syslog-ng OSE can collect process accounting logs on Linux systems.Process accounting is the method of recording and summarizing commands executed on Linux, for example, the commands being run, the user who executed the command, CPU time used by the process, exit code, and so on. When process accounting (also called pacct) is enabled on a system, the kernel writes accounting records to the `/var/log/account/pacct` file (might vary between different Linux distributions).

To use the `pacct()` driver, the following conditions must be met:

- The syslog-ng OSE application must be compiled with the `--enable-pacct` option. Execute the `syslog-ng -V` command to check if your binary supports process accounting.
- The `pacctformat` plugin must be loaded. By default, syslog-ng OSE automatically loads the available modules.
- The `scl.conf` file must be included in your syslog-ng configuration:

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

- Process accounting must be running on the host. You can enable it with the `accton` command.

The `pacct()` driver parses the fields of the accounting logs and transforms them into name-value pairs. The fields are defined in the manual page of the accounting log file (`man acct`), syslog-ng OSE prepends every field with the `.pacct.` prefix. For example, the `ac_uid` field that contains the id of the user who started the process will be available under the `$.pacct.ac_uid` name. These can be used as macros in templates, in filters to select specific messages, and so on.

To use the `pacct()` driver, use the following syntax.

```
@version: 3.17
@include "scl.conf"
source s_pacct { pacct(); };
...
log { source(s_pacct); destination(...); };
```

The `pacct()` driver is actually a reusable configuration snippet configured to handle Linux accounting logs. For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find the source of the pacct configuration snippet on GitHub.

pacct() options

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

file()

Type:	filename with path
Default:	/var/log/account/pacct

Description: The file where the process accounting logs are stored — syslog-ng OSE reads accounting messages from this file.

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.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE stops.

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

`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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname`

flag by default.

- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng OSE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng OSE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${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 3.16 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.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string

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

Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
         )
      );
};
```

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 OSE).

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

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 OSE). 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. 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 OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 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.

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 OSE 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 OSE 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 OSE version 3.10 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 OSE discards these messages.
- The syslog-ng OSE application cannot resolve OIDs, you have to configure snmptrapd to do so. Note that because of a bug, if snmptrapd does not escape String values in the VarBindList if it can resolve an OID to a symbolic name. As a result, syslog-ng OSE cannot process traps that contain the = in the value of the string. To overcome this problem, disable resolving OIDs in snmptrapd. For details, see the documentation of snmptrapd.
- The colon (:) character is commonly used in SNMP traps. However, this character cannot be used in the name of syslog-ng OSE macros (name-value pairs). Therefore, the syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 OSE discards these messages.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE stops.

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
```

```
    );
};
```

persist-name()

Type:	string
Default:	

*Description:*If you receive the following error message during syslog-ng OSE 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")`.

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 OSE. 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 vs. 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 OSE leaves the ${MESSAGE} part empty. If set-message-macro() is set to yes, syslog-ng OSE generates a regular log message from the trap.

sun-streams: Collecting messages on Sun Solaris

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

NOTE:

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

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

NOTE:

Starting with version 3.7, the syslog-ng OSE system() driver automatically extracts the msgid from the message (if available), and stores it in the .solaris.msgid macro. To extract the msgid from the message without using the system()driver, use the extract-solaris-msgid() parser. You can find the exact source of this parser in the syslog-ng OSE GitHub repository.

Declaration:

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

Example: Using the sun-streams() driver

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

sun-streams() source options

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

door()

Type:	string
Default:	none

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

flags()

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

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 3.16 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.

hook-commands()

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

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE).

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections, otherwise `log-iw-size()` is divided by 10 (the default value of the `max-connections()` option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 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.

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

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

syslog() source options

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

encoding()

Type:	string
Default:	

Description: Specifies the character set (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 3.16 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 OSE.

  NOTE:

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

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.

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

NOTE: The syslog-ng OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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-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 OSE 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 OSE 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-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 OSE 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 OSE 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 OSE. For details on using name resolution in syslog-ng OSE, see Using name resolution in syslog-ng.
- s

NOTE:

If the log message does not contain a hostname in its `HOST` field, syslog-ng OSE 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 OSE 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 OSE server and also on every relay, otherwise syslog-ng OSE 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 OSE).

listen-backlog()

Type:	integer
Default:	256

Description: Available only for stream based transports (`unix-stream`, `tcp`, `tls`). In TCP, connections are treated incomplete until the three-way handshake is completed between the server and the client. Incomplete connection requests wait on the TCP port for the listener to accept the request. The `listen-backlog()` option sets the maximum number of incomplete connection requests. For example:

```
source  s_network  {
       network(
             ip("192.168.1.1")
             transport("tcp")
             listen-backlog(2048)
             );
};
```

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections, otherwise `log-iw-size()` is divided by 10 (the default value of the `max-connections()` option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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: 601 In case of UDP transport: 514

Description: The port number to bind to.

program-override()

Type:	string
Default:	

Description: Replaces the ${PROGRAM} part of the message with the parameter string. For example, to mark every message coming from the kernel, include the `program-override ("kernel")` option in the source containing `/proc/kmsg`.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make syslog-ng send messages to a broadcast address. For details, see the `socket(7)` manual page.

so-keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. For details, see the `socket(7)` manual page.

so-rcvbuf()

Type:	number
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 OSE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

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

so-sndbuf()

Type:	number
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 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.

transport()

Type:	udp, tcp, or tls
Default:	tcp

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

CAUTION:

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

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

tls()

Type:	tls options
Default:	n/a

Description: This option sets various options related to TLS encryption, for example, key/certificate files and trusted CA locations. TLS can be used only with tcp-based transport protocols. For details, see 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 OSE application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all

hosts which may get to syslog-ng are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

`system`: Collecting the system-specific log messages of a platform

Starting with version 3.2, syslog-ng OSE 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 OSE configuration file, syslog-ng OSE automatically adds the following sources to the syslog-ng OSE configuration.

NOTE:

syslog-ng OSE versions 3.2-3.3 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 OSE.

The `system()` driver is also used in the default configuration file of syslog-ng OSE. For details on the default configuration file, see Example: The default configuration file of syslog-ng OSE. Starting with syslog-ng OSE 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 OSE does not recognize the platform it is installed on, it does not add any sources.

Starting with version 3.6, syslog-ng OSE 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 8: Sources automatically added by syslog-ng Open Source Edition

Platform	Message source
AIX and Tru64	`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 OSE 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 OSE 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 OSE is running in a jail or a Linux Container (LXC), it will not read from the `/dev/kmsg` or `/proc/kmsg` files.

Solaris 8	`sun-streams("/dev/log");` NOTE: Starting with version 3.7, the syslog-ng OSE `system()` driver automatically extracts the msgid from the message (if available), and stores it in the `.solaris.msgid` macro. To extract the msgid from the message without using the `system()`driver, use the `extract-solaris-msgid()` parser. You can find the exact source of this parser in the syslog-ng OSE GitHub repository.

Platform	Message source
Solaris 9	`sun-streams("/dev/log" door("/etc/.syslog_door"));` NOTE: Starting with version 3.7, the syslog-ng OSE `system()` driver automatically extracts the msgid from the message (if available), and stores it in the `.solaris.msgid` macro. To extract the msgid from the message without using the `system()`driver, use the `extract-solaris-msgid()` parser. You can find the exact source of this parser in the syslog-ng OSE GitHub repository.
Solaris 10	`sun-streams("/dev/log" door("/var/run/syslog_door"));` NOTE: Starting with version 3.7, the syslog-ng OSE `system()` driver automatically extracts the msgid from the message (if available), and stores it in the `.solaris.msgid` macro. To extract the msgid from the message without using the `system()`driver, use the `extract-solaris-msgid()` parser. You can find the exact source of this parser in the syslog-ng OSE GitHub repository.

system() source options

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

exclude-kmsg()

Type:	yes | no
Default:	no

Description: This option makes it possible to avoid duplicate collection of kernel logs or errors in kernel log collection (for example, in scenarios where the log management on the host system and the containerized solution are collecting the kernel logs simultaneously). When set to `yes`, syslog-ng OSE will omit kernel logs on platforms where they are available separately. These logs will not be collected or forwarded, but you will be able to find them if you log in. Currently these platforms are the following:

- Linux without systemd
- FreeBSD

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

`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 OSE 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 OSE will revert to using `systemd-journal()` instead.

CAUTION:

Only one `systemd-journal()` source can be configured in the configuration file. If there are more than one `systemd-journal()` sources configured, syslog-ng OSE 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: 3.17

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: 3.17

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: 3.17

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 OSE application makes this information available in the following 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 OSE 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.

default-level()

Type:	string
Default:	notice

Description: The default level value if the PRIORITY entry does not exist.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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-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 OSE will retain the hostname information read from the systemd journal messages.
- If disabled (`keep-hostname(no)`), syslog-ng OSE 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.

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 OSE adds the `.journald.` prefix to every value.

read-old-records()

Type:	yes|no
Default:	yes

Description: If set to `yes`, syslog-ng OSE 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 OSE 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 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.

use-fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

`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 OSE 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: 3.17

source s_systemdd {
    systemd-syslog();
};

destination d_network {
    syslog("server.host");
};

log {
    source(s_systemdd);
```

```
    destination(d_network);
};
```

systemd-syslog() source options

The `systemd-syslog()` driver has the following options:

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()	
Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE stops.

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

`tcp`, `tcp6`, `udp`, `udp6`: Collecting messages from remote hosts using the BSD syslog protocol— 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.

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')
             )
       );
 };
```

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 credentials and other metadata

Starting with syslog-ng OSE 3.6, the `unix-stream()` and `unix-dgram()` sources automatically extract the available UNIX credentials and other metainformation from the received log messages. The syslog-ng OSE application can extract the following information on Linux and FreeBSD platforms (examples show the value of the macro for the `su - myuser` command). Similar information is available for the systemd-journal source.

Table 9: UNIX credentials available via UNIX domain sockets

Macro	Description
`${.unix.cmdline}`	The name (without the path) and command-line options of the executable belonging to the PID that sent the message. For example, `su - myuser`
`${.unix.exe}`	The path of the executable belonging to the PID that sent the message. For example, `/usr/bin/su`
`${.unix.gid}`	The group ID (GID) corresponding to the UID of the application that sent the log message. Note that this is the ID number of the group, not its human-readable name. For example, `0`
`${.unix.pid}`	The process ID (PID) of the application that sent the log message. For example, `774`. Note that on every UNIX platforms, if the `system()` source uses sockets, it will overwrite the PID macro with the value of `${.unix.pid}`, if it is available.
`${.unix.uid}`	The user ID (UID) of the application that sent the log message. Note that this is the ID number of the user, not its human-readable name. For example, `0`

unix-stream() and unix-dgram() source options

These two drivers behave similarly: they open an `AF_UNIX` socket and start listening on it for messages. The following options can be specified for these drivers:

create-dirs()

Type:	yes or no
Default:	no

Description: Enable creating non-existing directories when creating the socket files.

encoding()

Type:	string
Default:	

Description: Specifies the character set (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE`

priority if not specified otherwise.

- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng OSE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng OSE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${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 3.16 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.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

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

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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-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 OSE).

listen-backlog()

Type:	integer
Default:	256

Description: Available only for stream based transports (`unix-stream`, `tcp`, `tls`). In TCP, connections are treated incomplete until the three-way handshake is completed between the server and the client. Incomplete connection requests wait on the TCP port for the listener to accept the request. The `listen-backlog()` option sets the maximum number of incomplete connection requests. For example:

```
source  s_network  {
       network(
             ip("192.168.1.1")
             transport("tcp")
             listen-backlog(2048)
             );
};
```

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
Default:	100

Description: The size of the initial window, this value is used during flow control. If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections, otherwise `log-iw-size()` is divided by 10 (the default value of the `max-connections()` option). The resulting number is the initial window size of each connection. For optimal performance when receiving messages from syslog-ng OSE 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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 OSE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

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

tags()

Type:	string
Default:	

Description: Label the messages received from the source with custom tags. Tags must be unique, and enclosed between double quotes. When adding multiple tags, separate them with comma, for example `tags("dmz", "router")`. This option is available only in syslog-ng 3.1 and later.

time-zone()

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

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

The timezone can be specified 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.

`stdin`: Collecting messages from the standard input stream

The `stdin()` driver collects messages from the standard input stream. When the standard input stream is closed, syslog-ng stops and `stdin()` inherits all options from the `file()` source, including multi-line options, or `flags(no-parse)`.

The `stdin()` driver causes syslog-ng to exit once it hits end-of-file (EOF).

Declaration:

```
stdin();
```

Example: Using the stdin() driver

```
@version: 3.17
log {
       source { stdin(); };
       destination { file("/dev/stdout"); };
};
```

The following code snippet is an example of how the `stdin()` driver is used to collect a test message:

```
$ echo "this is a test message" | ./syslog-ng -Fe --no-caps
[2017-11-14T13:47:16.757938] syslog-ng starting up; version='3.12.1'
[2017-11-14T13:47:16.758195] syslog-ng shutting down; version='3.12.1'
Nov 14 13:47:16 testserver this is a test message
```

stdin() source options

The `stdin()` 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 character set (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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.
- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE 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 3.16 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

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

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.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE).

log-fetch-limit()

Type:	number
Default:	100

Description: The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if `log-fetch-limit()` is too high.

log-iw-size()

Type:	number
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`.

Description: Maximum length of a message in bytes. This length includes the entire message (the data structure and individual fields). The maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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 OSE 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 OSE 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 OSE 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 OSE application provides the following methods to process multi-line messages:

- 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 OSE 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 OSE 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.

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. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad-size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. 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 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.

destination: Forward, send, and store log messages

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 OSE 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 log: Filter and route log messages using log paths, flags, and filters.**

The following table lists the destination drivers available in syslog-ng OSE. If these destinations do not satisfy your needs, you can extend syslog-ng OSE and write your own destination, for example, in C, Java, or Python. For details, see Write your own custom destination in Java or Python.

Table 10: Destination drivers available in syslog-ng

Name	Description
amqp()	Publishes messages using the AMQP (Advanced Message Queuing Protocol).
elasticsearch() and elasticsearch2	Sends messages to an Elasticsearch server. The `elasticsearch2` driver supports Elasticsearch version 2 and newer.
file()	Writes messages to the specified file.
graphite()	Sends metrics to a Graphite server to store numeric time-series data.
graylog2()	Sends syslog messages to Graylog.
hdfs()	Sends messages into a file on a Hadoop Distributed File System (HDFS) node.
http()	Sends messages over the HTTP protocol. There are two different implementations of this driver: a Java-based http driver, and an http driver without Java.
kafka()	Publishes log messages to the Apache Kafka message bus, where subscribers can access them.
loggly()	Sends log messages to the Loggly Logging-as-a-Service provider.
logmatic()	Sends log messages to the Logmatic.io Logging-as-a-Service provider.
mongodb()	Sends messages to a MongoDB database.
network()	Sends messages to a remote host using the BSD-syslog protocol over

Name	Description
	IPv4 and IPv6. Supports the TCP, UDP, and TLS network protocols.
logstore()	Writes messages to the specified binary logstore file.
pipe()	Writes messages to the specified named pipe.
program()	Forks and launches the specified program, and sends messages to its standard input.
redis()	Sends messages as name-value pairs to a Redis key-value store.
riemann()	Sends metrics or events to a Riemann monitoring system.
smtp()	Sends e-mail messages to the specified recipients.
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 Installing syslog-ng.
snmp()	Sends messages to the specified remote host using the SNMP v2c or v3 protocol.
stomp()	Sends messages to a STOMP server.
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.
telegram()	Sends log messages to Telegram, which is a secure, cloud-based mobile and desktop messaging app.
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.

`amqp`: Publishing messages using AMQP

The `amqp()` driver publishes messages using the AMQP (Advanced Message Queuing Protocol). syslog-ng OSE supports AMQP versions 0.9.1 and 1.0. The syslog-ng OSE `amqp()` driver supports persistence, and every available exchange types.

The name-value pairs selected with the `value-pairs()` option will be sent as AMQP headers, while the body of the AMQP message is empty by default (but you can add custom content using the `body()` option). Publishing the name-value pairs as headers makes it possible to use the Headers exchange-type and subscribe only to interesting log streams. This solution is more flexible than using the `routing-key()` option.

For the list of available parameters, see amqp() destination options.

Declaration:

```
amqp( host("<amqp-server-address>") );
```

Example: Using the amqp() driver

The following example shows the default values of the available options.

```
destination d_amqp {
       amqp(
              vhost("/")
              host("127.0.0.1")
              port(5672)
              exchange("syslog")
              exchange-type("fanout")
              routing-key("")
              body("")
              persistent(yes)
              value-pairs(
                     scope("selected-macros" "nv-pairs" "sdata")
              )
       );
};
```

amqp() destination options

The `amqp()` driver publishes messages using the AMQP (Advanced Message Queuing Protocol).

The `amqp()` destination has the following options:

body()

Type:	string
Default:	empty string

Description: The body of the AMQP message. You can also use macros and templates.

ca-file()

Type:	string
Default:	N/A

Description: Name of a file, that contains the trusted CA certificate in PEM format. For example: `ca-file("/home/certs/syslog-ng/tls/cacert.pem")`. The syslog-ng OSE application uses this CA certificate to validate the certificate of the peer.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

Declaration:

```
destination   d_ampqp  {
       amqp(
              host("127.0.0.1")
              port(5672)
              username("test")
              password("test")
              tls(
                     ca-file("ca")
                     cert-file("cert")
                     key-file("key")
                     peer-verify(yes|no)
              )
       );
};
```

Make sure that you specify TLS options either using their own dedicated option (`ca-file()`, `cert-file()`, `key-file()`, and `peer-verify()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE application uses this certificate to authenticate the syslog-ng OSE 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 into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

Declaration:

```
destination   d_ampqp  {
       amqp(
              host("127.0.0.1")
              port(5672)
              username("test")
              password("test")
              tls(
                     ca-file("ca")
                     cert-file("cert")
                     key-file("key")
                     peer-verify(yes|no)
              )
       );
};
```

Make sure that you specify TLS options either using their own dedicated option (`ca-file()`, `cert-file()`, `key-file()`, and `peer-verify()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()	
Type:	string

Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

exchange()

Type:	string
Default:	syslog

Description: The name of the AMQP exchange where syslog-ng OSE sends the message. Exchanges take a message and route it into zero or more queues.

exchange-declare()

Type:	yes|no
Default:	no

Description: By default, syslog-ng OSE does not create non-existing exchanges. Use the `exchange-declare(yes)` option to automatically create exchanges.

exchange-type()

Type:	direct|fanout|topic|headers
Default:	fanout

Description: The type of the AMQP exchange.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()

Type:	string

Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE stops.

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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:	hostname or IP address
Default:	127.0.0.1

Description: The hostname or IP address of the AMQP server.

key-file()

Accepted values:	Filename
Default:	none

Description: The name of a file that contains an unencrypted private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng OSE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng OSE client on the destination server.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

Declaration:

```
destination   d_ampqp  {
       amqp(
              host("127.0.0.1")
              port(5672)
              username("test")
              password("test")
              tls(
                     ca-file("ca")
                     cert-file("cert")
                     key-file("key")
                     peer-verify(yes|no)
              )
       );
};
```

Make sure that you specify TLS options either using their own dedicated option (`ca-file()`, `cert-file()`, `key-file()`, and `peer-verify()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

password()

Type:	string
Default:	n/a

Description: The password used to authenticate on the AMQP server.

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 OSE will reject the connection.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

Declaration:

```
destination  d_ampqp {
    amqp(
        host("127.0.0.1")
        port(5672)
        username("test")
        password("test")
        tls(
            ca-file("ca")
            cert-file("cert")
            key-file("key")
            peer-verify(yes|no)
        )
```

```
    );
};
```

Make sure that you specify TLS options either using their own dedicated option (`ca-file()`, `cert-file()`, `key-file()`, and `peer-verify()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

persistent()

Type:	yes|no
Default:	yes

Description: If this option is enabled, the AMQP server or broker will store the messages on its hard disk. That way, the messages will be retained if the AMQP server is restarted, if the message queue is set to be durable on the AMQP server.

port()

Type:	number
Default:	5672

Description: The port number of the AMQP server.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE could not send a message, it will try again until the number of attempts reaches `retries`, then drops the message.

routing-key()

Type:	string
Default:	empty string

Description: Specifies a routing key for the exchange. The routing key selects certain messages published to an exchange to be routed to the bound queue. In other words, the routing key acts like a filter. The routing key can include macros and templates.

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.

username()

Type:	string
Default:	empty string

Description: The username used to authenticate on the AMQP server.

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.

vhost()

Type:	string
Default:	/

Description: The name of the AMQP virtual host to send the messages to.

`elasticsearch`: Sending messages directly to Elasticsearch version 1.x

Starting with version 3.7 of syslog-ng OSE can directly send log messages to Elasticsearch, allowing you to search and analyze your data in real time, and visualize it with Kibana.

Note the following limitations when using the syslog-ng OSE `elasticsearch` destination:

- This destination is only supported on the Linux platform.
- Since syslog-ng OSE uses the official Java Elasticsearch libraries, the `elasticsearch` destination has significant memory usage.
- Sending messages over the HTTP REST API is supported only using the `elastic2()` destination. Note that in HTTP mode, the `elasticsearch2` destination can send log messages to Elasticsearch version 1.x and newer. For details, see elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher.
- The log messages of the underlying client libraries are available in the `internal()` source of syslog-ng OSE.

Declaration:

```
@module mod-java
@include "scl.conf"

elasticsearch(
      index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
      type("test")
      cluster("syslog-ng")
);
```

Example: Sending log data to Elasticsearch version 1.x

The following example defines an `elasticsearch` destination that sends messages in transport mode to an Elasticsearch server version 1.x running on the localhost, using only the required parameters.

```
@module mod-java
@include "scl.conf"

destination d_elastic {
   elasticsearch(
      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 {
  elasticsearch(
    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);
};
```

- To install the software required for the `elasticsearch` destination, see Prerequisites.
- For details on how the `elasticsearch` destination works, see How syslog-ng OSE interacts with Elasticsearch.
- For the list of options, see Elasticsearch destination options.

The `elasticsearch()` driver is actually a reusable configuration snippet configured to receive log messages using the Java language-binding of syslog-ng OSE. 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 OSE 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

To send messages from syslog-ng OSE to Elasticsearch, complete the following steps.

Steps:

1. If you want to use the Java-based modules of syslog-ng OSE (for example, the Elasticsearch, HDFS, or Kafka destinations), you must compile syslog-ng OSE with Java support.
    - Download and install the Java Runtime Environment (JRE), 1.7 (or newer). You can use OpenJDK or Oracle JDK, other implementations are not tested.
    - Install gradle version 2.2.1 or newer.
    - Set `LD_LIBRARY_PATH` to include the `libjvm.so` file, for example:`LD_LIBRARY_PATH=/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/amd64/server:$LD_LIBRARY_PATH`

      Note that many platforms have a simplified links for Java libraries. Use the simplified path if available. If you use a startup script to start syslog-ng OSE set `LD_LIBRARY_PATH` in the script as well.
    - If you are behind an HTTP proxy, create a `gradle.properties` under the `modules/java-modules/` directory. Set the proxy parameters in the file. For details, see The Gradle User Guide.
2. Download the Elasticsearch libraries version 1.5 or newer from the 1.x line from https://www.elastic.co/downloads/elasticsearch. To use Elasticsearch 2.x or newer, use the `elasticsearch2()` destination (see elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher).
3. Extract the Elasticsearch libraries into a temporary directory, then collect the various `.jar` files into a single directory (for example, `/opt/elasticsearch/lib/`) where syslog-ng OSE can access them. You must specify this directory in the syslog-ng OSE configuration file. The files are located in the `lib` directory and its subdirectories of the Elasticsearch release package.

How syslog-ng OSE interacts with Elasticsearch

The syslog-ng OSE application sends the log messages to the official Elasticsearch client library, which forwards the data to the Elasticsearch nodes. The way syslog-ng OSE interacts with Elasticsearch is described in the following steps.

- After syslog-ng OSE is started and the first message arrives to the `elasticsearch` destination, the `elasticsearch` destination tries to connect to the Elasticsearch server or cluster. If the connection fails, syslog-ng OSE will repeatedly attempt to connect again after the period set in `time-reopen()` expires.
- If the connection is established, syslog-ng OSE sends JSON-formatted messages to Elasticsearch.
    - If `flush-limit` is set to 1: syslog-ng OSE sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, syslog-ng OSE repeats sending the message, as set in the `retries()`

parameter. If sending the message fails for `retries()` times, syslog-ng OSE 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 OSE sends messages in a batch, and receives the response asynchronously. In case of a problem, syslog-ng OSE 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 OSE recognizes the error. In node mode, about 1000 messages can be lost.

- If `concurrent-requests` is higher than 1, syslog-ng OSE 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().

Client modes

The syslog-ng OSE application can interact with Elasticsearch in transport mode or node mode.

- **Transport mode**

  The syslog-ng OSE application uses the transport client API of Elasticsearch, and uses the `server()`, `port()`, and `cluster()` options from the syslog-ng OSE configuration file.

- **Node mode**

  The syslog-ng OSE application acts as an Elasticsearch node (client no-data), using the node client API of Elasticsearch. Further options for the node can be describe in an Elasticsearch configuration file specified in the `resource()` option.

  NOTE:

  In Node mode, it is required to define the home of the elasticsearch installation with the `path.home` paramter in the `.yml` file. For example: `path.home: /usr/share/elasticsearch`.

Elasticsearch destination options

The `elasticsearch` 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 `elasticsearch` destination has the following options.

Required options:

The following options are required: `index()`, `type()`. In node mode, the `cluster()` and the `resource()` options are required as well. Note that to use `elasticsearch`, you must add the following lines to the beginning of your syslog-ng OSE configuration:

```
@module mod-java
@include "scl.conf"
```

client-lib-dir()

Type:	string
Default:	The syslog-ng OSE 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 OSE configuration (for example, because you have multiple Java-based destinations), syslog-ng OSE will merge every available paths to a single list.

For the `elasticsearch` destination, include the path to the directory where you copied the required libraries (see Prerequisites), for example, `client-lib-dir ("/opt/elasticsearch/libs")`.

client-mode()

Type:	transport | node
Default:	node

Description: Specifies the client mode used to connect to the Elasticsearch server, for example, `client-mode("node")`.

- **Transport mode**

  The syslog-ng OSE application uses the transport client API of Elasticsearch, and uses the `server()`, `port()`, and `cluster()` options from the syslog-ng OSE configuration file.

- **Node mode**

  The syslog-ng OSE 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`.

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)`

In version 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE 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")
        cluster-url("http://node01:9200 http://node02:9200")
    );
};
```

concurrent-requests()

Type:	number
Default:	0

Description: The number of concurrent (simultaneous) requests that syslog-ng OSE 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 unaccessible.

custom-id()

Type:	template or template function
Default:	N/A

Description: Use this option to specify a custom ID for the records inserted into Elasticsearch. If this option is not set, the Elasticsearch server automatically generates and ID for the message. For example: `custom-id(${UNIQID})` (Note that to use the `${UNIQID}` macro, the `use-uniqid()` global option must be enabled. For details, see use-uniqid().)

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()

Type:	yes|no
Default:	no

Description: If set to `yes`, syslog-ng OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string

Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

flush-limit()

Type:	number
Default:	5000

Description: The number of messages that syslog-ng OSE sends to the Elasticsearch server in a single batch.

- If `flush-limit` is set to 1: syslog-ng OSE sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, syslog-ng OSE repeats sending the message, as set in the `retries()` parameter. If sending the message fails for `retries()` times, syslog-ng OSE 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 OSE sends messages in a batch, and receives the response asynchronously. In case of a problem, syslog-ng OSE 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 OSE recognizes the error. In node mode, about 1000 messages can be lost.

- If `concurrent-requests` is higher than 1, syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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()

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}")`.

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng OSE 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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `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 OSE attempts to send a message to this destination. If syslog-ng OSE could not send a message, it will try again until the number of attempts reaches `retries`, then drops the message.

resource()

Type:	string
Default:	N/A

Description: The list of Elasticsearch resources to load, separated by semicolons. For example, `resource ("/home/user/elasticsearch/elasticsearch.yml;/home/user/elasticsearch/elasticsearch2.yml")`.

server()

Type:	list of hostnames
Default:	127.0.0.1

Description: Specifies the hostname or IP address of the Elasticsearch server. When specifying an IP address, IPv4 (for example, `192.168.0.1`) or IPv6 (for example, `[::1]`) can be used as well. When specifying multiple addresses, use space to separate the addresses, for example, `server("127.0.0.1 remote-server-hostname1 remote-server-hostname2")`

This option is used only in transport mode: `client-mode("transport")`

template()

Type:	template or template function
Default:	$(format-json --scope rfc5424 --exclude DATE --key ISODATE @timestamp-p=${ISODATE})

Description: The message as sent to the Elasticsearch server. Typically, you will want to use the command-line notation of the `format-json` template function.

To add a `@timestamp` field to the message, for example, to use with Kibana, include the `@timestamp=${ISODATE}` expression in the template. For example: `template($(format-json --scope rfc5424 --exclude DATE --key ISODATE @timestamp=${ISODATE}))`

For details on formatting messages in JSON format, see 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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.

type()

Type:	string
Default:	N/A

Description: The type of the index. For example, `type("test")`.

elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher

Starting with version 3.7 of syslog-ng OSE can directly send log messages to Elasticsearch, allowing you to search and analyze your data in real time, and visualize it with Kibana.

Note the following limitations when using the syslog-ng OSE `elasticsearch2` destination:

- This destination is only supported on the Linux platform.
- Since syslog-ng OSE 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 OSE.

Declaration:

```
@module mod-java
@include "scl.conf"

elasticsearch2(
      index("syslog-ng")
      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")
            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")
        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);
};
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the syslog-ng OSE 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("&amp;lt;path-to-your-java-
keystore&amp;gt;.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("&amp;lt;path-to-your-java-
keystore&amp;gt;.jks")
              java-truststore-password("password-to-your-keystore")
       );
 };
```

- To install the software required for the `elasticsearch2` destination, see Prerequisites.
- For details on how the `elasticsearch2` destination works, see How syslog-ng OSE interacts with Elasticsearch.
- For the list of options, see Elasticsearch2 destination options.

The `elasticsearch2()` driver is actually a reusable configuration snippet configured to receive log messages using the Java language-binding of syslog-ng OSE. 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 OSE 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

To send messages from syslog-ng OSE to Elasticsearch, complete the following steps.

Steps:

1. Download and install the Java Runtime Environment (JRE), 2.x (or newer). The syslog-ng OSE `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.

   NOTE:

   This step is only required if you use the `elasticsearch2` destination in node mode or transport mode.

2. Download the Elasticsearch libraries (version 2.x or newer from the 2.x line) from https://www.elastic.co/downloads/elasticsearch.

   NOTE:

   This step is only required if you use the `elasticsearch2` destination in node mode or transport mode.

3. Extract the Elasticsearch libraries into a temporary directory, then collect the various `.jar` files into a single directory (for example, `/opt/elasticsearch/lib/`) where syslog-ng OSE can access them. You must specify this directory in the syslog-ng OSE configuration file. The files are located in the `lib` directory and its subdirectories of the Elasticsearch release package.

How syslog-ng OSE interacts with Elasticsearch

The syslog-ng OSE application sends the log messages to the official Elasticsearch client library, which forwards the data to the Elasticsearch nodes. The way syslog-ng OSE interacts with Elasticsearch is described in the following steps.

- After syslog-ng OSE 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 OSE will repeatedly attempt to connect again after the period set in `time-reopen()` expires.
- If the connection is established, syslog-ng OSE sends JSON-formatted messages to Elasticsearch.
  - If `flush-limit` is set to 1: syslog-ng OSE sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, syslog-ng OSE repeats sending the message, as set in the `retries()`

parameter. If sending the message fails for `retries()` times, syslog-ng OSE 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 OSE sends messages in a batch, and receives the response asynchronously. In case of a problem, syslog-ng OSE 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 OSE recognizes the error. In node mode, about 1000 messages can be lost.

- If `concurrent-requests` is higher than 1, syslog-ng OSE 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 3.10 and newer of syslog-ng OSE automatically converts the timestamp (date) of the message to UTC, as needed by Elasticsearch and Kibana.

Client modes

The syslog-ng OSE application can interact with Elasticsearch in the following modes of operation: http, https, node, searchguard, and transport.

- **HTTP mode**

  The syslog-ng OSE application sends messages over HTTP using the REST API of Elasticsearch, and uses the `cluster-url()` and `cluster()` options from the syslog-ng OSE configuration file. In HTTP mode, syslog-ng OSE `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTP mode is available in syslog-ng OSE version 3.8 and newer.

  In version 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE does not have any direct influence on it.

- **HTTPS mode**

  The syslog-ng OSE 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 OSE configuration file. In HTTPS mode, syslog-ng OSE `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTPS mode is available in syslog-ng OSE version 3.10 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 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE does not have any direct influence on it.

- **Transport mode**

  The syslog-ng OSE application uses the transport client API of Elasticsearch, and uses the `server()`, `port()`, and `cluster()` options from the syslog-ng OSE configuration file.

- **Node mode**

  The syslog-ng OSE 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 from syslog-ng OSE to Elasticsearch 2.x. For Elasticsearch versions 5.x and newer, use HTTPS mode. For details on configuring Search Guard mode, see Search Guard and syslog-ng OSE.

Search Guard and syslog-ng OSE

Purpose:

Version 3.9 and later supports the Search Guard Elasticsearch plugin (version 2.4.1.16 and newer) to encrypt and authenticate your connections to from syslog-ng OSE to Elasticsearch 2 and newer. To configure syslog-ng OSE to send messages to an Elasticsearch 2.x cluster that uses Search Guard, complete the following steps.

To connect to an Elasticsearch 5.x or newer cluster, use HTTPS mode.

Steps:

1. Install the Search Guard plugin on your syslog-ng OSE host. Use the plugin version that matches the version of your Elasticsearch installation.

   ```
   sudo /usr/share/elasticsearch/bin/plugin install -b com.floragunn/search-guard-ssl/<version-number-of-the-plugin>
   ```

2. Create a certificate for your syslog-ng OSE host, and add the certificate to the `SYSLOG_NG-NODE_NAME-keystore.jks` file. You can configure the location of this file in the Elasticsearch resources file under the `path.conf` parameter. For details, see the Search Guard documentation.

3. Configure an Elasticsearch destination in syslog-ng OSE that uses the `searchguard` client mode. For example:

   ```
   destination d_elasticsearch {
       elasticsearch2(
          client-lib-dir("/usr/share/elasticsearch/plugins/search-guard-ssl/*.jar:/usr/share/elasticsearch/lib")
          index("syslog-${YEAR}.${MONTH}.${DAY}")
          type("syslog")
          time-zone("UTC")
          client-mode("searchguard")
          resource("/etc/syslog-ng/elasticsearch.yml")
       );
    };
   ```

4. Configure the Elasticsearch resource file (for example, `/etc/syslog-ng/elasticsearch.yml`) as needed for your environment. Note the `searchguard:` section.

   ```
   cluster:
     name: elasticsearch
   discovery:
     zen:
       ping:
         unicast:
           hosts:
             - <ip-address-of-the-elasticsearch-server>
   node:
     name: syslog_ng_secure
     data; false
     master: false
   path:
     home: /etc/syslog-ng
     conf: /etc/syslog-ng
   searchguard:
     ssl:
       transport:
         keystore_filepath: syslog_ng-keystore.jks
   ```

```
keystore_password: changeit
truststore_filepath: truststore.jks
truststore_password: changeit
enforce_hostname_verification: true
```

Elasticsearch2 destination options

The `elasticsearch2` destination can directly send log messages to Elasticsearch, allowing you to search and analyze your data in real time, and visualize it with Kibana. The `elasticsearch2` destination has the following options.

Required options:

The following options are required: `index()`, `type()`. In node mode, either the `cluster()` or the `resource()` option is required as well. Note that to use `elasticsearch2`, you must add the following lines to the beginning of your syslog-ng OSE configuration:

```
@module mod-java
@include "scl.conf"
```

client-lib-dir()

Type:	string
Default:	The syslog-ng OSE 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 OSE configuration (for example, because you have multiple Java-based destinations), syslog-ng OSE 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 OSE application sends messages over HTTP using the REST API of Elasticsearch, and uses the `cluster-url()` and `cluster()` options from the syslog-ng OSE configuration file. In HTTP mode, syslog-ng OSE `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTP mode is available in syslog-ng OSE version 3.8 and newer.

  In version 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE does not have any direct influence on it.

- **HTTPS mode**

  The syslog-ng OSE 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 OSE configuration file. In HTTPS mode, syslog-ng OSE `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTPS mode is available in syslog-ng OSE version 3.10 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 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE does not have any direct influence on it.

- **Transport mode**

  The syslog-ng OSE application uses the transport client API of Elasticsearch, and uses the `server()`, `port()`, and `cluster()` options from the syslog-ng OSE configuration file.

- **Node mode**

  The syslog-ng OSE 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 from syslog-ng OSE to Elasticsearch 2.x. For Elasticsearch versions 5.x

and newer, use HTTPS mode. For details on configuring Search Guard mode, see Search Guard and syslog-ng OSE.

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)`

In version 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE 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")
       cluster-url("http://node01:9200 http://node02:9200")
    );
};
```

concurrent-requests()

Type:	number
Default:	0

Description: The number of concurrent (simultaneous) requests that syslog-ng OSE 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 unaccessible.

custom-id()

Type:	template or template function
Default:	N/A

Description: Use this option to specify a custom ID for the records inserted into Elasticsearch. If this option is not set, the Elasticsearch server automatically generates and ID for the message. For example: `custom-id(${UNIQID})` (Note that to use the `${UNIQID}` macro, the `use-uniqid()` global option must be enabled. For details, see use-uniqid().)

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()

Type:	yes|no
Default:	no

Description: If set to `yes`, syslog-ng OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

flush-limit()

Type:	number
Default:	5000

Description: The number of messages that syslog-ng OSE sends to the Elasticsearch server in a single batch.

- If `flush-limit` is set to 1: syslog-ng OSE sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, syslog-ng OSE repeats sending the message, as set in the `retries()` parameter. If sending the message fails for `retries()` times, syslog-ng OSE 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 OSE sends messages in a batch, and receives the response asynchronously. In case of a problem, syslog-ng OSE 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 OSE recognizes the error. In node mode, about 1000 messages can be lost.
- If `concurrent-requests` is higher than 1, syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

http-auth-type()

Type:	none | basic | clientcert
Default:	none

Description: Determines how syslog-ng OSE 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 OSE version 3.10 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 OSE 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("&amp;lt;path-to-your-java-keystore&amp;gt;.jks")
        java-keystore-password("password-to-your-keystore")
        java-truststore-filepath("&amp;lt;path-to-your-java-
keystore&amp;gt;.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 OSE version 3.10 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 OSE version 3.10 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.

java-keystore-filepath()

Type:	string
Default:	N/A

Description: Path to the Java keystore file that stores the certificate that syslog-ng OSE 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 OSE version 3.10 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 OSE 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("&amp;lt;path-to-your-java-keystore&amp;gt;.jks")
             java-keystore-password("password-to-your-keystore")
             java-truststore-filepath("&amp;lt;path-to-your-java-
keystore&amp;gt;.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 OSE version 3.10 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 OSE 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("&amp;lt;path-to-your-java-keystore&amp;gt;.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("&amp;lt;path-to-your-java-
keystore&amp;gt;.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 OSE 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 OSE 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 OSE version 3.10 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 OSE 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("&amp;lt;path-to-your-java-keystore&amp;gt;.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("&amp;lt;path-to-your-java-
keystore&amp;gt;.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 OSE version 3.10 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 OSE 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("&amp;lt;path-to-your-java-keystore&amp;gt;.jks")
              java-keystore-password("password-to-your-keystore")
              java-truststore-filepath("&amp;lt;path-to-your-java-
keystore&amp;gt;.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 OSE 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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `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 OSE attempts to send a message to this destination. If syslog-ng OSE 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 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The syslog-ng OSE application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), syslog-ng OSE 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 OSE checks the state of the cluster. If the primary shards of the cluster are not active, syslog-ng OSE 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

Version 3.10 and newer of syslog-ng OSE 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().

> 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.

type()

Type:	string
Default:	N/A

Description: The type of the index. For example, `type("test")`.

Example use cases of sending logs to Elasticsearch using syslog-ng

This section aims to give you some practical examples about how to make the most of your Elasticsearch-based logging using syslog-ng. Read the following blog posts to learn how to:

- Parse data with syslog-ng, store in Elasticsearch, and analyze with Kibana
- Get started on Red Hat Enterprise Linux / CentOS using Elasticsearch 6 and syslog-ng
- Send netdata metrics through syslog-ng to Elasticsearch
- Visualize your data using:
  - heat maps

    This example uses the GeoIP2 parser. For details about the GeoIP2 parser, see Looking up GeoIP2 data from IP addresses.
  - time lapse videos

`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 OSE 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 OSE.

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 OSE using the `syslog-ng-ctl reload` command, or use another method to send a SIGHUP to syslog-ng OSE.

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.

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 3.16, the default value of this option is -1, so syslog-ng OSE 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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

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 OSE.

  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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

fsync()

Type:	yes or no
Default:	no

Description: Forces an `fsync()` call on the destination fd after each write. Note: enabling this option may seriously degrade performance.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 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.

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 OSE 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 OSE.

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 OSE 3.4 and later.

overwrite-if-older()

Type:	number (seconds)
Default:	0

Description: If set to a value higher than 0, syslog-ng OSE 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 OSE 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 OSE. 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 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.

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.

`graphite`: Sending metrics to Graphite

The graphite() destination can send metrics to a Graphite server to store numeric time-series data. There are many ways to feed the Graphite template function with name value pairs. The syslog-ng OSE CSV and PatternDB parsers (for details, see Using pattern parsers) can parse log messages and generate name value pairs based on message content. The CSV parser (for details, see Parsing messages with comma-separated and similar values) can be used for logs which have a constant field based structure, like the Apache web server access logs. The patterndb parser can parse information and can extract important fields from free form log messages, as long as patterns describing the log messages are available. Another way is to send JSON-based log messages (for details, see The JSON parser) to syslog-ng OSE, like running a simple shell script collecting metrics and running it from cron regularly.

To see an example of how the graphite() destination is used to collect statistics coming from syslog-ng, see the blog post Collecting syslog-ng statistics to Graphite.

Declaration:

```
graphite(payload());
```

Example: Using the graphite() driver

To use the graphite() destination, the only mandatory parameter is payload, which specifies the value pairs to send to graphite. In the following example any value pairs starting with "monitor." are forwarded to graphite.

```
destination d_graphite { graphite(payload("--key monitor.*")); };
```

NOTE:

The graphite() destination is only a wrapper around the network() destination and the graphite-output template function. If you want to fine-tune the TCP parameters, use the network() destination instead, as described in graphite-output.

graphite() destination options

The graphite() destination has the following options:

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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:	hostname or IP address
Default:	localhost

Description: The hostname or IP address of the Graphite server.

port()

Type:	number
Default:	2003

Description: The port number of the Graphite server.

payload()

Type:	parameter list of the `payload()` option
Default:	`empty string`

Description: The payload() option allows you to select which value pairs to forward to graphite.

The syntax of `payload` is different from the syntax of `value-pairs()`: use the command-line syntax used in the format-json template function. For details on using the payload() option, see graphite-output.

NOTE:

If left empty, there is no data to be forwarded to Graphite.

Sending logs to Graylog

`graylog2()`: Sending logs to Graylog

You can use the `graylog2()` destination and a Graylog Extended Log Format (GELF) template to send syslog messages to Graylog.

You can forward simple name-value pairs where the name starts with a dot or underscore. If names of your name-value pairs include dots other than the first character, you should use JSON formatting directly instead of the GELF template and send logs to a raw TCP port in Graylog, which can then extract fields from nested JSON.

Declaration:

```
graylog2();
```

Example: Using the graylog2() driver

You can send syslog messages to Graylog using the `graylog2()` destination. The `graylog2()` destination uses the GELF template, the native data format of Graylog.

1. On the Graylog side, configure a GELF TCP input. For more information, see the relevant Graylog documentation.
2. On the syslog-ng side, configure the name or IP address of the host running Graylog.

   ```
   destination d_graylog {
       graylog2(
           host("172.16.146.142")
       );
    };
   ```

   If you parsed your messages using syslog-ng, the template also forwards any name-value pairs where the name starts with a dot or underscore.

NOTE:

If there is a dot in a field name other than the first character, syslog-ng creates nested JSON while formatting the message. Nested JSON is not automatically parsed in GELF messages.

Sending nested JSON to Graylog

While sending nested JSON inside GELF is possible, it is not convenient. If you use parsing and normalization in syslog-ng and dot notation in field names, use pure JSON instead of GELF to forward your messages.

1. On the Graylog side, create a new raw TCP input.
2. Still in Graylog, once the raw TCP input is ready, add a JSON extractor to it.
3. On the syslog-ng side, use a network destination combined with a template utilizing format-json as shown in the example below:

```
destination d_jsontcp {
    network(
        "172.16.146.142"
        port("5555")
        transport(tcp)
        template("$(format-json --scope all-nv-pairs)\n")
    );
};
```

graylog2() destination options

The `graylog2()` destination has the following options:

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
```

```
    );
};
```

hdfs: Storing messages on the Hadoop Distributed File System (HDFS)

Starting with version 3.7, syslog-ng OSE 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 amounts 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).

For more information about the benefits of using syslog-ng as a data collection, processing, and filtering tool in a Hadoop environment, see the blog post Filling your data lake with log messages: the syslog-ng Hadoop (HDFS) destination.

Note the following limitations when using the syslog-ng OSE `hdfs` destination:

- This destination is only supported on the Linux platform.
- Since syslog-ng OSE uses the official Java HDFS client, the `hdfs` destination has significant memory usage (about 400MB).
- 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 OSE application to influence when the messages are actually written to disk. This means that syslog-ng OSE cannot guarantee that a message sent to HDFS is actually written to disk. When using flow-control, syslog-ng OSE 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 OSE.

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 OSE interacts with HDFS.
- For details on using MapR-FS, see Storing messages with MapR-FS.
- For details on using Kerberos authentication, see Kerberos authentication with syslog-ng hdfs() destination.
- 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 OSE. 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 OSE 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

To send messages from syslog-ng OSE to HDFS, complete the following steps.

Steps:

1. If you want to use the Java-based modules of syslog-ng OSE (for example, the Elasticsearch, HDFS, or Kafka destinations), you must compile syslog-ng OSE with Java support.
    - Download and install the Java Runtime Environment (JRE), 1.7 (or newer). You can use OpenJDK or Oracle JDK, other implementations are not tested.
    - Install gradle version 2.2.1 or newer.
    - Set `LD_LIBRARY_PATH` to include the `libjvm.so` file, for example:`LD_LIBRARY_PATH=/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/amd64/server:$LD_LIBRARY_PATH`

      Note that many platforms have a simplified links for Java libraries. Use the simplified path if available. If you use a startup script to start syslog-ng OSE set `LD_LIBRARY_PATH` in the script as well.
    - If you are behind an HTTP proxy, create a `gradle.properties` under the `modules/java-modules/` directory. Set the proxy parameters in the file. For details, see The Gradle User Guide.
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 temporary directory, then collect the various `.jar` files into a single directory (for example, `/opt/hadoop/lib/`) where syslog-ng OSE can access them. You must specify this directory in the syslog-ng OSE configuration file. The files are located in the various `lib` directories under the `share/` directory of the Hadoop release package. (For example, in Hadoop 2.7, required files are `common/hadoop-common-2.7.0.jar`, `common/libs/*.jar`, `hdfs/hadoop-hdfs-2.7.0.jar`, `hdfs/lib/*`, but this may change between Hadoop releases, so it is easier to copy every `.jar` file into a single directory.

How syslog-ng OSE interacts with HDFS

The syslog-ng OSE application sends the log messages to the official HDFS client library, which forwards the data to the HDFS nodes. The way syslog-ng OSE interacts with HDFS is described in the following steps.

1. After syslog-ng OSE 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 OSE will repeatedly attempt to connect again after the period set in `time-reopen()` expires.
2. syslog-ng OSE checks if the path to the logfile exists. If a directory does not exist syslog-ng OSE automatically creates it. syslog-ng OSE creates the destination file (using the filename set in the syslog-ng OSE 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 OSE will write all incoming messages into the `hdfs` destination.

NOTE:

When the `hdfs-append-enabled()` option is set to `true`, syslog-ng OSE 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 OSE application to influence when the messages are actually written to disk. This means that syslog-ng OSE cannot guarantee that a message sent to HDFS is actually written to disk. When using flow-control, syslog-ng OSE 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 OSE 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 OSE drops the message.
4. The syslog-ng OSE application closes the destination file in the following cases:
   - syslog-ng OSE is reloaded
   - syslog-ng OSE is restarted
   - The HDFS client returns an error.
5. If the file is closed and you have set an archive directory, syslog-ng OSE moves the file to this directory. If syslog-ng OSE cannot move the file for some reason (for example, syslog-ng OSE cannot connect to the HDFS NameNode), the file remains at its original location, syslog-ng OSE will not try to move it again.

Storing messages with MapR-FS

The syslog-ng OSE application is also compatible with MapR File System (MapR-FS). 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 OSE, 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 OSE.

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 3.10 and later supports Kerberos authentication to authenticate the connection to your Hadoop cluster. syslog-ng OSE 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 OSE 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 OSE 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 OSE. For details, see the Kerberos documentation.
- You have installed and configured the Kerberos client packages on the host running syslog-ng OSE. (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 OSE configuration:

```
@module mod-java
@include "scl.conf"
```

client-lib-dir()

Type:	string
Default:	The syslog-ng OSE 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 OSE configuration (for example, because you have multiple Java-based destinations), syslog-ng OSE 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/*.jar:/opt/hadoop/libs/*.jar")`.

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
};
```

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp

of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

hdfs-append-enabled()

Type:	true | false
Default:	false

Description: When `hdfs-append-enabled` is set to `true`, syslog-ng OSE will append new data to the end of an already existing HDFS file.

When `hdfs-append-enabled` is set to `false`, the syslog-ng OSE 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 OSE 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 OSE 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 OSE will move the closed log files. If syslog-ng OSE cannot move the file for some reason (for example, syslog-ng OSE cannot connect to the HDFS NameNode), the file remains at its original location. For example, `hdfs-archive-dir ("/usr/hdfs/archive/")`.

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 OSE checks if the path to the logfile exists. If a directory does not exist syslog-ng OSE automatically creates it.

`hdfs-file()` supports the usage of macros. This means that syslog-ng OSE 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 OSE 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 OSE 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 OSE 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")`.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE stops.

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng OSE 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 OSE version 3.10 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 OSE version 3.10 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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `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 OSE attempts to send a message to this destination. If syslog-ng OSE 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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.

Posting messages over HTTP

Version 3.7 of syslog-ng OSE can directly post log messages to web services using the HTTP protocol. Error and status messages received from the HTTP server are forwarded to the internal logs of syslog-ng OSE. The current implementation has the following limitations:

- This destination is only supported on the Linux platform.
- Only HTTP connections are supported, HTTPS is not.
- This destination requires Java. For an `http` destination that does not use Java, see http: Posting messages over HTTP without Java.

Declaration:

```
@module  mod-java

java(
      class-path("/syslog-ng/install_dir/lib/syslog-ng/java-modules/*.jar")
      class-name("org.syslog_ng.http.HTTPDestination")

      option("url",  "http://<server-address>:<port-number>")

);
```

Example: Sending log data to a web service

The following example defines an `http` destination.

```
@module mod-java

destination d_http {
    java(
        class-path("/syslog-ng/install_dir/lib/syslog-ng/java-modules/*.jar")
        class-name("org.syslog_ng.http.HTTPDestination")

        option("url", "http://192.168.1.1:80")
    );
};
```

```
log
    { source(s_file); destination(d_http); flags(flow-control); };
```

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.

HTTP destination options

The `http` destination of syslog-ng OSE can directly post log messages to web services using the HTTP protocol. The `http` destination has the following options. Some of these options are directly used by the Java code underlying the `http` destination, therefore these options must be specified in the following format:

`option("<option-name>", "<option-value>")`

For example, `option("url", "http://<server-address>:<port-number>")`. The exact format to use is indicated in the description of the option.

Required options:

The following options are required: `url()`. Note that to use `http`, you must add the following line to the beginning of your syslog-ng OSE configuration:

```
@module mod-java
```

class-name()

Type:	string
Default:	N/A

Description: The name of the class (including the name of the package) that includes the destination driver to use.

For the `http` destination, use this option as `class-name("org.syslog-ng.http.HTTPDestination")`.

client-lib-dir()

Type:	string
Default:	The syslog-ng OSE 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 OSE configuration (for example, because you have multiple Java-based destinations), syslog-ng OSE will merge every available paths to a single list.

For the `http` destination, include the path to the java modules of syslog-ng OSE, for example, `class-path("/syslog-ng/install_dir/lib/syslog-ng/java-modules/*.jar")`.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string

Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng OSE 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.

method()

Type:	DELETE | HEAD | GET | OPTIONS | POST | PUT | TRACE
Default:	PUT

Description: Specifies the HTTP method to use when sending the message to the server. Available in syslog-ng OSE version 3.7.2 and newer.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

url()

Type:	URL
Default:	

Description: Specifies the hostname or IP address and optionally the port number of the web service that can receive log data via HTTP. Use a colon (`:`) after the address to specify the port number of the server. You can also use macros, templates, and template functions in the URL, for example: `http://host.example.com:8080/${MACRO1}/${MACRO2}/script")`

`http`: Posting messages over HTTP without Java

Version 3.8 of syslog-ng OSE can directly post log messages to web services using the HTTP protocol, without having to use Java. The current implementation has the following limitations:

- Only the PUT and the POST methods are supported.

HTTPS connection, as well as password- and certificate-based authentication is supported.

Example: Client certificate authentication with HTTPS

```
destination d_https {
    http(
       [...]
       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); };
```

You can also use the http() destination to forward log messages to Splunk using syslog-ng OSE.

HTTP destination options

The `http` destination of syslog-ng OSE can directly post log messages to web services using the HTTP protocol. The `http` destination has the following options.

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.

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 OSE 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 into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
               )
       );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE 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 into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
               )
       );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE application uses this certificate to authenticate the syslog-ng OSE 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 into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
            )
    );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that syslog-ng OSE 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 OSE. 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 $1}' | xargs
echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

Note that starting with version 3.10, when syslog-ng OSE receives TLS-encrypted connections, the order of ciphers set on the syslog-ng OSE server takes precedence over the client settings.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
               )
       );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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: The name of a file that contains an unencrypted private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng OSE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng OSE client on the destination server.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
               )
       );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE 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 OSE will reject the connection.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
```

```
            )
        );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

persist-name()

Type:	string
Default:	

*Description:*If you receive the following error message during syslog-ng OSE 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")`.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE 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 OSE will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, syslog-ng OSE 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`.

An alternative way to specify this option is to put into a `tls()` block and specify it there, together with any other TLS options. This allows you to separate these options and ensure better readability.

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>)
               )
       );
};
```

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. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

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 OSE. 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 disk-buffer 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 3.11 and later.

url()

Type:	URL
Default:	http://localhost/

Description: Specifies the hostname or IP address and optionally the port number of the web service that can receive log data via HTTP. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://127.0.0.1:8000`

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 OSE uses to authenticate on the server where it sends the messages.

kafka: Publishing messages to Apache Kafka

Starting with version 3.7, syslog-ng OSE can directly publish log messages to the Apache Kafka message bus, where subscribers can access them.

- This destination is only supported on the Linux platform.
- Since syslog-ng OSE 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 OSE.

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 OSE 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 OSE. 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 OSE 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

To publish messages from syslog-ng OSE to Apache Kafka, complete the following steps.

Steps:

1. If you want to use the Java-based modules of syslog-ng OSE (for example, the Elasticsearch, HDFS, or Kafka destinations), you must compile syslog-ng OSE with Java support.
    - Download and install the Java Runtime Environment (JRE), 1.7 (or newer). You can use OpenJDK or Oracle JDK, other implementations are not tested.
    - Install gradle version 2.2.1 or newer.
    - Set `LD_LIBRARY_PATH` to include the `libjvm.so` file, for example:`LD_LIBRARY_PATH=/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/amd64/server:$LD_LIBRARY_PATH`

      Note that many platforms have a simplified links for Java libraries. Use the simplified path if available. If you use a startup script to start syslog-ng OSE set `LD_LIBRARY_PATH` in the script as well.
    - If you are behind an HTTP proxy, create a `gradle.properties` under the `modules/java-modules/` directory. Set the proxy parameters in the file. For details, see The Gradle User Guide.
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 OSE can access them. You must specify this directory in the syslog-ng OSE configuration file.
4. Check if the following files in the Kafka libraries have the same version number: `slf4j-api-<version-number>.jar`, `slf4j-log4j12-<version-number>.jar`. If the version

number of these files is different, complete the following steps:

a. Delete one of the files (for example, `slf4j-log4j12-<version-number>.jar`).

b. Download a version that matches the version number of the other file (for example, 1.7.6) from the official SLF4J distribution.

c. Copy the downloaded file into the directory of your Kafka library files (for example, `/opt/kafka/lib/`).

How syslog-ng OSE interacts with Apache Kafka

When stopping the syslog-ng OSE application, syslog-ng OSE 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 OSE 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 OSE `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 OSE 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 `kafka`, you must add the following lines to the beginning of your syslog-ng OSE configuration:

```
@module mod-java
@include "scl.conf"
```

client-lib-dir()

Type:	string
Default:	The syslog-ng OSE 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 OSE configuration (for example, because you have multiple Java-based destinations), syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng OSE 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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `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 OSE application reads this file and passes the properties to the Kafka Producer. If a property is defined both in the syslog-ng OSE configuration file (`syslog-ng.conf`) and in the properties file, then syslog-ng OSE uses the definition from the syslog-ng OSE configuration file.

The syslog-ng OSE `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 acknowledgement method and compression:

```
acks=all
compression.type=snappy
```

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE 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 OSE sends the message reliably: it sends a message to the Kafka server, then waits for a reply. In case of failure, syslog-ng OSE repeats sending the message, as set in the `retries()` parameter. If sending the message fails for `retries()` times, syslog-ng OSE drops the message.

This method ensures reliable message transfer, but is very slow.

When `sync-send` is set to `false`, syslog-ng OSE sends messages asynchronously, and receives the response asynchronously. In case of a problem, syslog-ng OSE cannot resend the messages.

This method is fast, but the transfer is not reliable. Several thousands of messages can be lost before syslog-ng OSE 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 disk-buffer 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 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.

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.

loggly: Using Loggly

The loggly() destination sends log messages to the Loggly Logging-as-a-Service provider. You can send log messages over TCP, or encrypted with TLS.

Declaration:

```
loggly(token());
```

Example: Using the loggly() driver

To use the loggly() destination, the only mandatory parameter is your user token. The following example sends every log from the system() source to your Loggly account.

```
log {
      source { system(); };
      destination { loggly(token("<USER-TOKEN-AS-PROVIDED-BY-LOGGLY>")); };
};
```

The following example uses TLS encryption. Before using it, download the CA certificate of Loggly and copy it to your hosts (for example, into the /etc/ssl/certs/ directory.

```
log {
      destination {
            loggly(token("<USER-TOKEN-AS-PROVIDED-BY-LOGGLY>") port(6514)
                  tls(peer-verify(required-trusted) ca-dir('/etc/ssl/certs'))
            );
      };
};
```

The following example parses the access logs of an Apache webserver from a file and sends them to Loggly in JSON format.

```
log {
      source { file("/var/log/apache2/access.log" flags(no-parse)); };
      parser { apache-accesslog-parser(); };
      destination {
            loggly(token("<USER-TOKEN-AS-PROVIDED-BY-LOGGLY>")
                tag(apache)
                template("$(format-json .apache.* timestamp=${ISODATE})"));
      };
}
```

To use the loggly() driver, the scl.conf file must be included in your syslog-ng OSE configuration:

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

The `loggly()` 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.

loggly() destination options

The `loggly()` destination has the following options. You can also set other options of the underlying `tcp()` driver (for example, port number or TLS-encryption).

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

token()

Type:	string
Default:	

Description: Your Customer Token that you received from Loggly.

`logmatic`: Using Logmatic.io

The `logmatic()` destination sends log messages to the Logmatic.io Logging-as-a-Service provider. You can send log messages over TCP, or encrypted with TLS.

Declaration:

```
logmatic(token());
```

Example: Using the logmatic() driver

To use the logmatic() destination, the only mandatory parameter is your user token. The following example sends every log from the `system()` source to your Logmatic.io account.

```
log {
       source { system(); };
       destination { logmatic(token("<API-KEY-AS-PROVIDED-BY-LOGMATIC.IO>"));
};
 };
```

The following example uses TLS encryption. Before using it, download the CA certificate of Logmatic.io and copy it to your hosts (for example, into the `/etc/ssl/certs/` directory.

```
log {
       destination {
             logmatic(token("<API-KEY-AS-PROVIDED-BY-LOGMATIC.IO>") port(6514)
                   tls(peer-verify(required-trusted) ca-dir('/etc/ssl/certs'))
             );
       };
 };
```

The following example parses the access logs of an Apache webserver from a file and sends them to Logmatic.io in JSON format.

```
log {
       source { file("/var/log/apache2/access.log" flags(no-parse)); };
       parser { apache-accesslog-parser(); };
       destination {
             logmatic(token("<API-KEY-AS-PROVIDED-BY-LOGMATIC.IO>")
                 tag(apache)
                 template("$(format-json .apache.* timestamp=${ISODATE})"));
       };
 }
```

To use the `logmatic()` driver, the `scl.conf` file must be included in your syslog-ng OSE configuration:

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

The `logmatic()` 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.

logmatic() destination options

The `logmatic()` destination has the following options. You can also set other options of the underlying `tcp()` driver (for example, port number or TLS-encryption).

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

token()

Type:	string
Default:	

Description: Your API Key that you received from Logmatic.io.

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.

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 OSE, use the key() parameter of the value-pairs() option.

The syslog-ng OSE mongodb() driver is compatible with MongoDB server version 1.4 and newer.

NOTE:

By default, syslog-ng OSE 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 (as used in syslog-ng OSE version 3.7), 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 OSE connects the MongoDB server

When syslog-ng OSE connects the MongoDB server during startup, it completes the following steps.

1. The syslog-ng OSE 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 OSE 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 OSE 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 OSE connects the received address.
3. When syslog-ng OSE 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 OSE 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 OSE configuration file, otherwise you risk losing log messages if all the addresses listed in the syslog-ng OSE configuration are offline.**
- **Addresses retrieved from the MongoDB servers are not stored, and can be lost when syslog-ng OSE 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 OSE 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 OSE cannot insert a log message into the database because of an error.

   In such case, syslog-ng OSE 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 OSE 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.

The `mongodb()` destination has the following options:

collection()

Type:	string
Default:	messages

Description: The name of the MongoDB collection where the log messages are stored (collections are similar to SQL tables). Note that the name of the collection must not start with a dollar sign ($), and that it may contain dot (.) characters.

CAUTION:

Hazard of data loss! The syslog-ng OSE application does not verify that the specified collection name does not contain invalid characters. If you specify a collection with an invalid name, the log messages sent to the MongoDB database will be irrevocably lost without any warning.

database() (DEPRECATED)

Type:	string
Default:	syslog

This option is deprecated and will be removed from syslog-ng OSE. Use the `uri()` option instead.

Description: The name of the MongoDB database where the log messages are stored. Note that the name of the database must not start with a dollar sign ($) and it cannot contain dot (.) characters.

CAUTION:

Hazard of data loss! The syslog-ng OSE application does not verify that the specified database name does not contain invalid characters. If you specify a database with an invalid name, the log messages sent to the MongoDB database will be irrevocably lost without any warning.

disk-buffer()

Description: This option enables putting outgoing messages into the disk buffer of the destination to avoid message loss in case of a system failure on the destination side. It has the following options:

reliable()	
Type:	yes|no
Default:	no

Description: If set to `yes`, syslog-ng OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
```

```
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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.

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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `drop-property`: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string`: Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message`: Drop the entire message silently, without logging the error.
- `silently-drop-property`: Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string`: Convert the property to string silently, without logging the error.

password() (DEPRECATED)

Type:	string
Default:	n/a

This option is deprecated and will be removed from syslog-ng OSE. Use the `uri()` option instead.

Description: Password of the database user.

path() (DEPRECATED)

Type:	string
Default:	empty

This option is deprecated and will be removed from syslog-ng OSE. Use the `uri()` option instead.

Description: If the `path()` option is set, syslog-ng OSE will connect to the database using the specified UNIX domain socket. Note that you cannot set the `path()` and the `servers()` options at the same time.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE 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 OSE uses a one-minute timeout: if an operation times out, syslog-ng OSE assumes the operation has failed.

safe-mode() (DEPRECATED)

Type:	yes or no
Default:	yes

This option is deprecated and will be removed from syslog-ng OSE. Use the `uri()` option instead.

Description: If `safe-mode()` is enabled, syslog-ng OSE performs an extra check after each insert to verify that the insert succeeded. The insert is successful only if this second check is successful. Note that enabling this option reduces the performance of the driver.

servers() (DEPRECATED)

Type:	list of hostname:port pairs
Default:	127.0.0.1:27017

This option is deprecated and will be removed from syslog-ng OSE. Use the `uri()` option instead.

Description: Specifies the hostname or IP address and the port number of the database server. When specifying an IP address, IPv4 (for example, `192.168.0.1`) or IPv6 (for example, `[::1]`) can be used as well.

To send the messages to a MongoDB replicaset, specify the addresses of the database servers as a comma-separated list, for example: `servers (192.168.1.1:27017,192.168.3.3:27017)`

For details on how syslog-ng OSE connects the MongoDB server, see How syslog-ng OSE connects the MongoDB server.

To connect to the server using a UNIX domain socket, use path option. Note that you cannot set the `path()` and the `servers()` options at the same time.

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.

uri()

Type:	string
Default:	mongodb://127.0.0.1:27017/syslog?wtimeoutMS=60000&socketTimeoutMS=60000&connectTimeoutMS=60000

Description: Available in syslog-ng OSE 3.8 and later. Please refer to the MongoDB URI format documentation for detailed syntax.

username() (DEPRECATED)

Type:	string
Default:	n/a

This option is deprecated and will be removed from syslog-ng OSE. Use the `uri()` option instead.

Description: Name of the database user. Note that the `mongodb()` driver currently does not support TLS-encrypted authentication.

value-pairs()

Type:	parameter list of the `value-pairs()` option
Default:	`scope("selected-macros" "nv-pairs")`

Description: The `value-pairs()` option creates structured name-value pairs from the data and metadata of the log message. For details on using `value-pairs()`, see Structuring macros, metadata, and other value-pairs.

NOTE:

Empty keys are not logged.

NOTE:

By default, syslog-ng OSE 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.

`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 OSE sends messages using the TCP protocol to port 601.

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, see BSD-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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.	

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")
              )
       );
 };
```

failover()

Description: Available only in syslog-ng Open Source Edition version 3.17 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 OSE 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 OSE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

failback()

Description: Available only in syslog-ng Open Source Edition version 3.17 and later.

When syslog-ng OSE 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 OSE behaves as follows:

- **round-robin mode**: If `failback()` is not set, syslog-ng OSE does not attempt to

return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

Example: round-robin mode

In the following example syslog-ng OSE 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 OSE attempts to return to the primary server.

  After syslog-ng OSE 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 OSE tries to connect the primary server using the last probe.

  NOTE: syslog-ng OSE 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 OSE 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.

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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE 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 OSE.

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 OSE 3.4 and later.

port() or destport()

Type:	number
Default:	601

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the SO_BROADCAST socket option required to make syslog-ng send messages to a broadcast address. 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.

CAUTION:

To use spoofing on Microsoft Windows platforms, you must also set the `spoof-interface()` option as well.

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 OSE 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 OSE version 3.4 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 OSE 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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 OSE 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().

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.

osquery: Sending log messages to osquery's syslog table

The `osquery()` driver sends log messages to osquery's syslog table.

The syslog table contains logs forwarded over a named pipe from syslog-ng. When an osquery process that supports the syslog table starts up, it creates (and properly sets permissions for) a named pipe for syslog-ng to write to.

Example: Using the osquery() destination driver

Run osqueryi:

```
osqueryi --enable_syslog
         --disable-events=false
```

To store the database on disk:

```
osqueryi --enable_syslog
         --disable-events=false
         --database_path=/tmp/osquery.db
```

To set up a custom named pipe:

```
osqueryi --enable_syslog
         --disable-events=false
         --database_path=/tmp/osquery.db
         --syslog_pipe_path=/tmp/osq.pipe
```

Example configuration:

```
@version: 3.12
@include "scl.conf"
```

```
source s_net {
  network(port(5514));
};

destination d_osquery {
  # custom pipe path:
  #osquery(pipe("/tmp/osq.pipe"));

  # backup outgoing logs:
  #osquery(file("/var/log/osquery_inserts.log" template(t_osquery)));

  # defaults
  osquery();
};

log {
 source(s_net);
 destination(d_osquery);
 flags(flow-control);
};
```

osquery() destination options

The `osquery()` destination has the following options:

file()

Type:	string
Default:	N/A

Description: Specifies a path to the file where log messages are stored, for example, for debug purposes.

Specifying this option is optional. However, when you start losing logs for some reason, then it is recommended to write outgoing log messages to a specified file, in the same format that messages are written to the pipe. You can also use a `template()` function called `t_osquery`, which re-formats messages so they comply with the text-based protocol that osquery accepts.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

pipe()

Type:	string
Default:	N/A

Description: Specifies a custom path to the named pipe that acts as the interface between osquery and syslog-ng. (The default path is set in the SCL file.)

Specifying this option is optional.

`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 OSE 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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

group()

Type:	string
Default:	Use the global settings

Description: Set the group of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: `group()`.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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.

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 OSE 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 OSE.

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 OSE 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 OSE 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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.

`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 OSE 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:

- The syslog-ng OSE 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 OSE to execute external applications.
- The syslog-ng OSE 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 OSE 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 OSE application stops, it will automatically stop the external program. To avoid restarting the application when syslog-ng OSE 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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the

message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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.

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 OSE). 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 OSE 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 OSE 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 OSE.

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 OSE 3.4 and later.

Note that in earlier versions of syslog-ng OSE, 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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.

pseudofile()

The `pseudofile()` destination driver is a very simple driver, aimed at delivering messages to special files such as files in the `/proc`, `/dev` or `/sys` directories. It opens and closes the file after each write operation, instead of keeping it open. It does not append further data. It does not support templates in the filename, and does not have a queue, processing is performed immediately as read by the source. Therefore, no loss is possible, but it takes CPU time from the source, so it is not adequate in high traffic situations.

Declaration:

```
pseudofile(filename options());
```

pseudofile() destination options

The `pseudofile()` destination has the following options:

file()

Type:	filename with path
Default:	

Description: The file to write messages to, including the path.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE. 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.

`redis`: Storing name-value pairs in Redis

The `redis()` driver sends messages as name-value pairs to a Redis key-value store.

For the list of available parameters, see redis() destination options.

Declaration:

```
redis(
      host("<redis-server-address>")
      port("<redis-server-port>")
      auth("<redis-server-password>") # Optional, for password-protected servers
      command("<redis-command>", "<first-command-parameter>", "<second-command-
parameter>", "<third-command-parameter>")
 );
```

Example: Using the redis() driver

The following destination counts the number of log messages received per host.

```
destination d_redis {
       redis(
              host("localhost")
              port(6379)
              command("HINCRBY", "hosts", "$HOST", "1")
       );
};
```

The following example creates a statistic from Apache webserver logs about the browsers that the visitors use (per minute)

```
@version: 3.17

source s_apache {
       file("/var/log/apache2/access.log");
};

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('""[]')
       );
};

destination d_redis {
       redis( command("HINCRBY" "${MONTH_ABBREV} ${DAY} ${HOUR}:${MIN}"
"${APACHE.USER_AGENT}" "1"));
};

log {
       source(s_apache);
       parser(p_apache);
       destination(d_redis);
};
```

redis() destination options

The `redis()` driver sends messages as name-value pairs to a Redis key-value store.

The `redis()` destination has the following options:

auth()

Type:	hostname or IP address
Default:	N/A

Description: The password used for authentication on a password-protected Redis server. Available in syslog-ng OSE version 3.10 and later.

command()

Type:	comma-separated list of strings ("<redis-command>", "<first-command-parameter>", "<second-command-parameter>", "<third-command-parameter>")
Default:	empty string

Description: The Redis command to execute, for example, LPUSH, INCR, or HINCRBY. Using the HINCRBY command with an increment value of 1 allows you to create various statistics. For example, the `command("HINCRBY" "${HOST}/programs" "${PROGRAM}" "1")` command counts the number of log messages on each host for each program.

Note the following points when using the `redis()` destination:

- You can use macros and templates in the parameters of the Redis command.
- Currently you can use only one command in a redis() destination.
- The syslog-ng OSE application ignores the return value of the command. If the Redis server returns an error, syslog-ng OSE closes the connection.

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a

faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
        )
    );
};
```

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string

Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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:	hostname or IP address
Default:	127.0.0.1

Description: The hostname or IP address of the Redis server.

port()

Type:	number
Default:	6379

Description: The port number of the Redis server.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE could not send a message, it will try again until the number of attempts reaches `retries`, then drops the message.

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.

`riemann`: Monitoring your data with Riemann

The `riemann()` driver sends your data (for example, metrics or events) to a Riemann monitoring system.

For the list of available parameters, see riemann() destination options.

Declaration:

```
riemann(
        server("<riemann-server-address>")
        port("<riemann-server-port>")
        metric("<the-metric-or-data-to-send-to-riemann>")
);
```

Example: Using the riemann() driver

The following destination sends the value of the SEQNUM macro (the number of messages sent to this destination) as a metric to the Riemann server.

```
@version: 3.17

source s_network {
        network(port(12345));
};

destination d_riemann {
        riemann(
                server("localhost")
                port(5555)
                ttl("300.5")
                metric(int("$SEQNUM"))
                description("syslog-ng riemann test")
                state("ok")
                attributes(x-ultimate-answer("$(+ $PID 42)")
                                key("MESSAGE", rekey(add-prefix("x-")) )
                                )
        );
};

log {
        source(s_network);
        destination(d_riemann);
        flags(flow-control);
};
```

For a detailed use-case on using syslog-ng OSE with the Riemann monitoring system, see the article A How to Guide on Modern Monitoring and Alerting by Fabien Wernli.

riemann() destination options

The `riemann()` driver sends metrics or events to a Riemann monitoring system.

The `riemann()` destination has the following options:

attributes()

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

Description: The `attributes()` option adds extra metadata to the Riemann event, that can be displayed on the Riemann dashboard. To specify the metadata to add, use the syntax of the `value-pairs()` option. For details on using `value-pairs()`, see Structuring macros, metadata, and other value-pairs.

description()

Type:	template, macro, or string
Default:	

Description: The value to add as the description field of the Riemann event.

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
};
```

event-time()

Type:	template, macro, or string
Default:	${UNIXTIME}

Description: Instead of the arrival time into Riemann, syslog-ng OSE can also send its own timestamp value.

This can be useful if Riemann is inaccessible for a while, and the messages are collected in the disk buffer until Riemann is accessible again. In this case, it would be difficult to differentiate between messages based on the arrival time only, because this would mean that there would be hundreds of messages with the same arrival time. This issue can be solved by using this option.

The `event-time()` option takes an optional parameter specifying whether the time format is in seconds or microseconds. For example:

```
event-time("$(* $UNIXTIME 1000000)" microseconds)
event-time("12345678" microseconds)
```

```
event-time("12345678" seconds)
event-time("12345678")
```

In case the parameter is omitted, syslog-ng OSE defaults to the seconds version. In case the `event-time()` option is omitted altogether, syslog-ng OSE defaults to the seconds version with `$UNIXTIME`.

Note that the time format parameter requires:

- riemann-c-client 1.10.0 or newer

  In older versions of riemann-c-client, the microseconds option is not available.

  In case your distribution does not contain a recent enough version of riemann-c-client and you wish to use microseconds, install a new version from .

  If you installed the new version in a custom location (instead of the default one), make sure that you append the directory of the pkg-config file (`.pc` file) to the environment variable `export PKG_CONFIG_PATH=...`.

  After calling `configure`, you should see the following message in the case of successful installation:

  ```
  [...]
    Riemann destination (module): yes, microseconds: yes
  [...]
  ```

- Riemann 2.13 or newer

  Older versions of Riemann cannot handle microseconds. No error will be indicated, however, the time of the event will be set to the timestamp when the message arrived to Riemann.

Example: Example event-time() option

```
destination d_riemann {
      riemann(
      server("127.0.0.1")
      port(5555)
      event-time("${UNIXTIME}")
      [...]
      );
};
```

flush-lines()

Type:	number
Default:	1

Description: The syslog-ng OSE application can send the messages in a batch to the Riemann server. To send messages in batches, increase the `flush-lines()` parameter (by

default, it is set to 1). The syslog-ng OSE 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. Note that currently the `riemann()` destination does not have a timeout for sending messages if the batch is not full.

For example, if you set `flush-lines()` to 100, syslog-ng OSE waits for 100 messages. If the source sends a few messages, but less than 100 messages, syslog-ng OSE will not send the messages to the destination. If you stop or reload syslog-ng OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

If an error occurs while sending the messages to the server, syslog-ng OSE will try to resend every message from the batch. If it does not succeed (you can set the number of retry attempts in the `retries()` option), syslog-ng OSE drops every message in the batch.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the

following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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, macro, or string
Default:	${HOST}

Description: The value to add as the host field of the Riemann event.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

metric()

Type:	template, macro, or string
Default:	

Description: The numeric value to add as the metric field of the Riemann event. If possible, include type-hinting as well, otherwise the Riemann server will interpret the value as a floating-point number. The following example specifies the SEQNUM macro as an integer.

```
metric(int("$SEQNUM"))
```

port()

Type:	number
Default:	5555

Description: The port number of the Riemann server.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE could not send a message, it will try again until the number of attempts reaches `retries`, then drops the message.

server()

Type:	hostname or IP address
Default:	127.0.0.1

Description: The hostname or IP address of the Riemann server.

service()

Type:	template, macro, or string
Default:	${PROGRAM}

Description: The value to add as the service field of the Riemann event.

state()

Type:	template, macro, or string
Default:	

Description: The value to add as the state field of the Riemann event.

tags()

Type:	string list
Default:	the tags already assigned to the message

Description: The list of tags to add as the tags field of the Riemann event. If not specified syslog-ng OSE automatically adds the tags already assigned to the message. If you set the `tags()` option, only the tags you specify will be added to the event.

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.

timeout()

Type:	number [seconds]
Default:	

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the Riemann server if exceeded. By default, the timeout is disabled.

ttl()

Type:	template, macro, or number
Default:	

Description: The value (in seconds) to add as the ttl (time-to-live) field of the Riemann event.

type()

Type:	tcp | tls | udp
Default:	tcp

Description: The type of the network connection to the Riemann server: TCP, TLS, or UDP. For TLS connections, set the `ca-file()` option to authenticate the Riemann server, and the `cert-file()` and `key-file()` options if the Riemann server requires authentication from its clients.

Declaration 1:

```
destination  d_riemann  {
        riemann(
                server("127.0.0.1")
                port(5672)
                type(
                     "tls"
                     ca-file("ca")
                     cert-file("cert")
                     key-file("key")
                )
        );
};
```

An alternative way to specify TLS options is to group them into a `tls()` block. This allows you to separate them and ensure better readability.

Declaration 2:

```
destination d_riemann {
       riemann(
               server("127.0.0.1")
               port(5672)
               type("tls")
               tls(
                       ca-file("ca")
                       cert-file("cert")
                       key-file("key")
               )
```

```
        );
};
```

Make sure that you specify TLS options either using `type()` or using the `tls()` block. Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

ca-file()

Type:	path to a CA certificate in PEM format
Default:	

Description: Path to the CA certificate in PEM format that signed the certificate of the Riemann server. When establishing TLS connection, syslog-ng OSE verifies the certificate of the Riemann server using this CA.

Alternative 1:

```
type(
        "tls"
        ca-file("/opt/syslog-ng/etc/syslog-ng/riemann-cacert.pem")
        )
```

Alternative 2:

```
riemann(
        .
        .
        type("tls")
         tls(
                ca-file("/opt/syslog-ng/etc/syslog-ng/riemann-cacert.pem")
        )
```

This option was called `cacert()` up until (and including) syslog-ng OSE version 3.12.

cert-file()

Type:	path to a certificate in PEM format
Default:	

Description: Path to the a certificate file in PEM format. When establishing TLS connection, syslog-ng OSE authenticates on the Riemann server using this certificate and the matching private key set in the `key-file()` option.

Note that you have to set the `cert-file()` and `key-file()` options only if the Riemann server requires authentication from the clients.

Alternative 1:

```
type(
        "tls"
         cert-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.pem")
```

```
        key-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.key")
        )
```

Alternative 2:

```
riemann(
        .
        .
        type("tls")
         tls(
               cert-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.pem")
               key-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.key")
        )
```

This option was called `cert()` in syslog-ng OSE version 3.7.

key-file()

Type:	path to a private key file
Default:	

Description: Path to the private key of the certificate file set in the `cert-file()` option. When establishing TLS connection, syslog-ng OSE authenticates on the Riemann server using this private key and the matching certificate set in the `cert-file()` option.

Note that you have to set the `cert-file()` and `key-file()` options only if the Riemann server requires authentication from the clients.

Alternative 1:

```
type(
        "tls"
         cert-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.pem")
        key-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.key")
        )
```

Alternative 2:

```
riemann(
        .
        .
        type("tls")
         tls(
               cert-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.pem")
               key-file("/opt/syslog-ng/etc/syslog-ng/riemann-client-cert.key")
        )
```

This option was called `key()` in syslog-ng OSE version 3.7.

smtp: Generating SMTP messages (e-mail) 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 e-mail in general. For example, syslog-ng OSE does not verify whether the recipient exists.

The `smtp()` driver sends e-mail messages triggered by log messages. The `smtp()` driver uses SMTP, without needing external applications. You can customize the main fields of the e-mail, add extra headers, send the e-mail to multiple recipients, and so on.

The `subject()`, `body()`, and `header()` fields may include macros which get expanded in the e-mail. For more information on available macros see Macros of syslog-ng OSE.

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:

The `smtp()` destination driver is available only in syslog-ng OSE 3.4 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 e-mail alerting with the `smtp()` driver

The following example sends an e-mail 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 e-mail 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 e-mail. You can also use macros in the string. Use `\n` to start a new line. For example:

```
body("syslog-ng OSE received the following alert from $HOST:\n$MSG")
```

bcc()

Type:	string
Default:	n/a

Description: The BCC recipient of the e-mail (contents of the BCC field). You can specify the e-mail address, or the name and the e-mail address. Set the `bcc()` option multiple times to send the e-mail 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 e-mail (contents of the CC field). You can specify the e-mail address, or the name and the e-mail address. Set the `cc()` option multiple times to send the e-mail 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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.	

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")
              )
       );
 };
```

from()

Type:	string
Default:	n/a

Description: The sender of the e-mail (contents of the FROM field). You can specify the e-mail address, or the name and the e-mail 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 e-mail 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 e-mail, other occurrences will be ignored.
- It is not possible to set the DATE, Return-Path, Original-Recipient, Content-*, MIME-*, Resent-*, Received headers.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string

Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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:	hostname or IP address
Default:	n/a

Description: Hostname or IP address of the SMTP server.

NOTE:

If you specify `host="localhost"`, syslog-ng OSE will use a socket to connect to the local SMTP server. Use `host="127.0.0.1"` to force TCP communication between syslog-ng OSE 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 e-mail address, or the name and the e-mail address. Set the `reply-to()` option multiple times to send the e-mail 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 OSE attempts to send a message to this destination. If syslog-ng OSE 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 e-mail. 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 disk-buffer 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 e-mail (contents of the TO field). You can specify the e-mail address, or the name and the e-mail address. Set the `to()` option multiple times to send the e-mail 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: Sending log messages to Splunk

Although syslog-ng OSE currently does not have any built-in integration with Splunk, the existing message-formatting features and flexibility of syslog-ng OSE allows you to forward your log messages to Splunk. In syslog-ng OSE version 3.8 or later, you can use the `http()` destination. In earlier versions, you can use the `program()` destination.

For details on forwarding log messages to Splunk with syslog-ng OSE see the following posts on the Splunk blog:

- syslog-ng and HEC: Scalable Aggregated Data Collection in Splunk
- Using Syslog-ng with Splunk

`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.

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 application requires read and write access to the SQL table, otherwise it cannot verify that the destination table exists.

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

NOTE:

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

The table and value parameters can include macros to create tables and columns dynamically (for details, see Macros of syslog-ng OSE).

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 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 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.

- The hostname of the database server is set 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 OSE will find the file: `ORACLE_BASE`, `ORACLE_HOME`, and `ORACLE_SID`. For details, see the documentation of the Oracle Instant Client.

- 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 OSE.

- 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 OSE, the types of database columns must be explicitly set for the Oracle destination. The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng, therefore it is usually recommended to use the `varchar2` or `clob` column type. (The maximum length of the messages can be set using the `log-msg-size()` option.) For details, see the following example.

- The Oracle Instant Client used by syslog-ng OSE 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 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 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 OSE, 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. 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 application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.

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

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

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

The way syslog-ng interacts with the database

Used SQL operations by syslog-ng

Create table:

- If the given table does not exist, syslog-ng tries to create it with the given column types.
- The syslog-ng OSE 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 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 tries to add columns in this table but never will delete or modify an existing column.
- If syslog-ng OSE cannot create or alter a table, it tries to do it again when reach the next `time-reopen()`.
- The syslog-ng OSE 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 OSE application automatically performs the escaping required to insert the messages into the database.
- If insert returns with error, syslog-ng tries to insert the message +two times by default, then drops it. Retrying time is the value of `time-reopen()`.

Encoding

The syslog-ng OSE application uses UTF-8 by default when writes logs into database.

Start/stop and reload

Start:

- The syslog-ng OSE application will connect to database automatically after starting regardless existing incoming messages.

Stop:

- The syslog-ng OSE application will close the connection to database before shutting down.

Possibility of losing logs:

- The syslog-ng OSE application cannot lose logs during shutting down if disk buffer was given and it is not full yet.
- The syslog-ng OSE application cannot lose logs during shutting down if disk buffer was not given.

Reload:

- The syslog-ng OSE 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 be grown continously.

MySQL-specific interaction methods

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

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 OSE 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`.

create-statement-append()

Type:	string
Default:	empty string

Description: Specifies additional SQL options that are appended to the CREATE statement. That way you can customize what happens when syslog-ng OSE creates a new table in the database. Consult the documentation of your database server for details on the available options. Syntax:

```
create-statement-append(<options-to-append>)
```

For example, you can appends the `ROW_FORMAT=COMPRESSED` option to MySQL create table statements:

```
create-statement-append(ROW_FORMAT=COMPRESSED)
```

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 OSE 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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
};
```

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 OSE from creating non-existing database tables automatically. The syslog-ng OSE application typically has to create tables if you use macros in the table names. Available in syslog-ng OSE version 3.2 and later.
- *explicit-commits*: By default, syslog-ng OSE 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 OSE version 3.2 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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the

message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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:	hostname or IP address
Default:	n/a

Description: Hostname of the database server. Note that Oracle destinations do not use this parameter, but retrieve the hostname from the `/etc/tnsnames.ora` file.

NOTE:

If you specify `host="localhost"`, syslog-ng will use a socket to connect to the local database server. Use `host="127.0.0.1"` to force TCP communication between syslog-ng and the local database server.

To specify the socket to use, set and export the `MYSQL_UNIX_PORT` environment variable, for example `MYSQL_UNIX_PORT=/var/lib/mysql/mysql.sock; export MYSQL_UNIX_PORT`.

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 OSE application will create the name of indexes automaticaly 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 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.

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 OSE 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 OSE 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 OSE has successfully connected to the database. For example:

```
session-statements("SET COLLATION_CONNECTION='utf8_general_ci'")
```

CAUTION:

The syslog-ng OSE 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 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.

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)
```

stomp: Publishing messages using STOMP

The stomp() driver sends messages to servers (message brokers) using the Simple (or Streaming) Text Oriented Message Protocol (STOMP), formerly known as TTMP. syslog-ng OSE supports version 1.0 of the STOMP protocol. The syslog-ng OSE stomp() driver supports persistence.

The name-value pairs selected with the value-pairs() option will be sent as STOMP headers, while the body of the STOMP message is empty by default (but you can add custom content using the body() option). Publishing the name-value pairs as headers makes it possible to use the Headers exchange-type and subscribe only to interesting log streams.

For the list of available parameters, see stomp() destination options.

Declaration:

```
stomp( host("<stomp-server-address>") );
```

Example: Using the stomp() driver

The following example shows the default values of the available options.

```
destination d_stomp {
       stomp(
              host("localhost")
              port(61613)
              destination("/topic/syslog")
              body("")                    # optional, empty by default
              persistent(yes)
              ack(no)
              username("user")        # optional, empty by default
              password("password") # optional, empty by default
              value-pairs(scope(selected-macros, nv-pairs, sdata))
       );
};
```

stomp() destination options

The `stomp()` driver publishes messages using the Simple (or Streaming) Text Oriented Message Protocol (STOMP).

The `stomp()` destination has the following options:

ack()

Type:	yes|no
Default:	no

Description: Request the STOMP server to acknowledge the receipt of the messages. If you enable this option, then after sending a message, syslog-ng OSE waits until the server confirms that it has received the message. This delay can seriously limit the performance of syslog-ng OSE if the message rate is high, and the server cannot acknowledge the messages fast enough.

body()

Type:	string
Default:	empty string

Description: The body of the STOMP message. You can also use macros and templates.

destination()

Type:	string
Default:	/topic/syslog

Description: The name of the destination (message queue) on the STOMP server. It can include macros and templates.

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
        )
    );
};
```

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.

teardown()

Type:	string
Default:	N/A

Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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:	hostname or IP address
Default:	127.0.0.1

Description: The hostname or IP address of the STOMP server.

password()

Type:	string
Default:	n/a

Description: The password used to authenticate on the STOMP server.

persistent()

Type:	yes|no
Default:	yes

Description: If this option is enabled, the STOMP server or broker will store the messages on its hard disk. That way, the messages will be retained if the STOMP server is restarted, if the message queue is set to be durable on the STOMP server.

port()

Type:	number
Default:	61613

Description: The port number of the STOMP server.

retries()

Type:	number (of attempts)
Default:	3

Description: The number of times syslog-ng OSE attempts to send a message to this destination. If syslog-ng OSE could not send a message, it will try again until the number of attempts reaches `retries`, then drops the message.

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.

username()

Type:	string
Default:	empty string

Description: The username used to authenticate on the STOMP server.

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.

`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 — 601, 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')
              )
       );
};
```

NOTE:

If a message uses the IETF-syslog format (RFC5424), only the text of the message can be customized (that is, the $MESSAGE part of the log), the structure of the header is fixed.

syslog() destination options

The `syslog()` driver sends messages to a remote host (for example a syslog-ng server or relay) on the local intranet or internet using the RFC5424 syslog protocol developed by IETF (for details about the protocol, see 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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
 };
```

failover()

Description: Available only in syslog-ng Open Source Edition version 3.17 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 OSE 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 OSE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

failback()

Description: Available only in syslog-ng Open Source Edition version 3.17 and later.

When syslog-ng OSE 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 OSE behaves as follows:

- **round-robin mode**: If failback() is not set, syslog-ng OSE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

  Example: round-robin mode

  In the following example syslog-ng OSE 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 OSE attempts to return to the primary server.

  After syslog-ng OSE 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 OSE tries to connect the primary server using the last probe.

  NOTE: syslog-ng OSE 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 OSE 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.

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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

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 OSE 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 OSE.

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 OSE 3.4 and later.

port() or destport()

Type:	number
Default:	601

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the SO_BROADCAST socket option required to make syslog-ng send messages to a broadcast address. 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.

CAUTION:

To use spoofing on Microsoft Windows platforms, you must also set the `spoof-interface()` option as well.

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 OSE 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 OSE version 3.4 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 OSE 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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 OSE 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().

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.

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 3.16 and later. The node that receives this message must use the default-network-drivers() source to properly handle the messages.

The follwing 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 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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()

Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.	

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")
             )
       );
 };
```

failover()

Description: Available only in syslog-ng Open Source Edition version 3.17 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 OSE 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 OSE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

CAUTION:

The failover servers must be accessible on the same port as the primary server.

failback()

Description: Available only in syslog-ng Open Source Edition version 3.17 and later.

When syslog-ng OSE 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 OSE behaves as follows:

- **round-robin mode**: If `failback()` is not set, syslog-ng OSE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

  Example: round-robin mode

  In the following example syslog-ng OSE 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 OSE attempts to return to the primary server.

  After syslog-ng OSE 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 OSE tries to connect the primary server using the last probe.

  NOTE: syslog-ng OSE 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 OSE 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.

(missing or bad snippet)

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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

ip-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 OSE 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 OSE.

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 OSE 3.4 and later.

port() or destport()

Type:	number
Default:	601

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

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 OSE 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 OSE version 3.4 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 OSE 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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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 OSE 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().

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.

`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')
              )
       );
};
```

`Telegram`: Sending messages to Telegram

The `telegram()` destination sends log messages to Telegram, which is a secure, cloud-based mobile and desktop messaging app.

Declaration:

```
telegram(parameters);
```

Example: Using the telegram() driver

The following example creates a `telegram()` destination.

```
destination  d_telegram  {
       telegram(
             template("$(urlencode\"${MSG}\")")
             throttle(1)
             parse-mode("markdown")
             disable-web-page-preview("true")
             bot-id("<bot  id>")
             chat-id("<chat  id>")
       );
 };
```

telegram() destination options

The `telegram()` destination has the following options:

bot-id()

Type:	number
Default:	N/A

Description: This is a required option. Specifies the token for the bot necessary to access the Telegram HTTP API.

chat-id()

Type:	number
Default:	N/A

Description: This is a required option. Specifies the ID of the chat of the telegram destination.

disable-web-page-preview()

Type:	boolean
Default:	true

Description: Disables link previews for links in the message. By default, the disable-web-page-preview value is `true`. From security point of view, One Identity recommends to leave it true, otherwise malicious messages can trick the telegram destination to generate traffic to any URL.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE starts.

shutdown()

Type:	string
Default:	N/A

Description: Defines the external program that is executed as syslog-ng OSE stops.

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the

following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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")
        )
     );
};
```

parse-mode()

Type:	string
Default:	none

Description: Formats the message in a markdown-style or HTML-style formatting. By default, the parse-mode value is `markdown`, which means that the message is formatted in markdown style.

template()

Type:	string
Default:	$(urlencode \"${MSG} \")

Description: Specifies the content of the message. For security reasons, One Identity recommends using `urlencode` in the template function.

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.

`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:

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 OSE cannot lose logs in case of reload/restart, unreachable destination or syslog-ng OSE crash. This solution provides a slower, but reliable disk-buffer option. It is created and initialized at startup and gradually grows as new messages arrive. If set to `no`, the normal disk-buffer will be used. This provides a faster, but less reliable disk-buffer option.

CAUTION:

Hazard of data loss! If you change the value of `reliable()` option when there are messages in the disk-buffer, the messages stored in the disk-buffer will be lost.

dir()	
Type:	string
Default:	N/A

Description: Defines the folder where the disk-buffer files are stored.

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 OSE 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 OSE will look for or create disk-buffer files in their old location. To ensure that syslog-ng OSE 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.

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")
              )
       );
};
```

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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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 OSE 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 OSE to execute external applications.

Using the hook-commands() when syslog-ng OSE starts or stops

To execute an external program when syslog-ng OSE starts or stops, use the following options:

startup()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE starts.	

shutdown()	
Type:	string
Default:	N/A
Description: Defines the external program that is executed as syslog-ng OSE stops.	

Using the hook-commands() when syslog-ng OSE reloads

To execute an external program when the syslog-ng OSE configuration is initiated or torn down, for example, on startup/shutdown or during a syslog-ng OSE reload, use the following options:

setup()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is initiated, for example, on startup or during a syslog-ng OSE reload.	

teardown()	
Type:	string
Default:	N/A
Description: Defines an external program that is executed when the syslog-ng OSE configuration is stopped or torn down, for example, on shutdown or during a syslog-ng OSE 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 as syslog-ng OSE 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 OSE 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.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is reloaded. Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the source.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make syslog-ng send messages to a broadcast address. For details, see the `socket(7)` manual page.

so-keepalive()

Type:	yes or no
Default:	no

Description: Enables keep-alive messages, keeping the socket open. This only effects TCP and UNIX-stream sockets. For details, see the `socket(7)` manual page.

mark-freq()

Accepted values:	number [seconds]
Default:	1200

Description: An alias for the obsolete `mark()` option, retained for compatibility with syslog-ng version 1.6.x. The number of seconds between two `MARK` messages. `MARK` messages are generated when there was no message traffic to inform the receiver that the connection is still alive. If set to zero (`0`), no `MARK` messages are sent. The `mark-freq()` can be set for global option and/or every `MARK` capable destination driver if `mark-mode()` is periodical or dst-idle or host-idle. If `mark-freq()` is not defined in the destination, then the `mark-freq()` will be inherited from the global options. If the destination uses internal `mark-mode()`, then the global `mark-freq()` will be valid (does not matter what `mark-freq()` set in the destination side).

mark-mode()

Accepted values:	`internal` | `dst-idle` | `host-idle` | `periodical` | `none` | `global`
Default:	`internal` for pipe, program drivers `none` for file, unix-dgram, unix-stream drivers `global` for syslog, tcp, udp destinations `host-idle` for global option

Description: The `mark-mode()` option can be set for the following destination drivers: file(), program(), unix-dgram(), unix-stream(), network(), pipe(), syslog() and in global option.

- `internal`: When internal mark mode is selected, internal source should be placed in the log path as this mode does not generate mark by itself at the destination. This mode only yields the mark messages from internal source. This is the mode as syslog-ng OSE 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 OSE.

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 OSE 3.4 and later.

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 OSE. 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 disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

Description: Convert timestamps to the timezone specified by this option. If this option is not set, then the original timezone information in the message is used. Converting the timezone changes the values of all date-related macros derived from the timestamp, for example, `HOUR`. For the complete list of such macros, see 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.

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.

`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"); };
```

Write your own custom destination in Java or Python

The syslog-ng OSE application is open source, so if you have the necessary programming skills, you can extend it if its features are not adequate for your particular environment or needs. You can write destinations and other extensions to syslog-ng OSE in C (the main language of syslog-ng OSE), or using its language bindings, for example, Java or Python. For details on extending syslog-ng OSE, see the syslog-ng OSE Developer 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.

Client-side failover

syslog-ng OSE 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 OSE switches to the third server in the list, and so on. If there are no more failover servers left, syslog-ng OSE 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 OSE nominates this destination over the failover servers, and handles it as the "primary" address.

When syslog-ng OSE 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 OSE behaves as follows:

- **round-robin mode**: If `failback()` is not set, syslog-ng OSE does not attempt to return to the primary server even if it becomes available. In case the failover server fails, syslog-ng OSE attempts to connect the next failover server in the list in round-robin fashion.

Example: round-robin mode

In the following example syslog-ng OSE 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 OSE attempts to return to the primary server.

  After syslog-ng OSE 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 OSE tries to connect the primary server using the last probe.

  NOTE: syslog-ng OSE 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 OSE 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 OSE is restarted, it attempts to connect the primary server.

If syslog-ng OSE uses TLS-encryption to communicate with the remote server, syslog-ng OSE 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 OSE 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.

log: Filter and route log messages using log paths, flags, and filters

Log paths
Managing incoming and outgoing messages with flow-control
Using disk-based and memory buffering
Filters
Dropping messages

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:

Declaration

```
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.

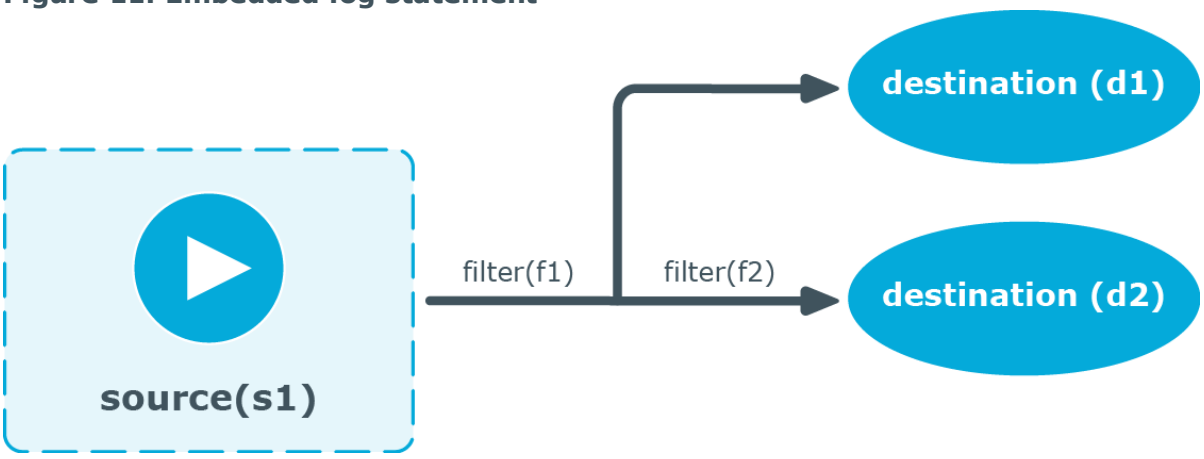
This default behavior can be changed using the `flags()` parameter. Flags apply to individual log paths, they are not global options. For details and examples on the available flags, see Log path flags. The effect and use of the `flow-control` flag is detailed in Managing incoming and outgoing messages 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.

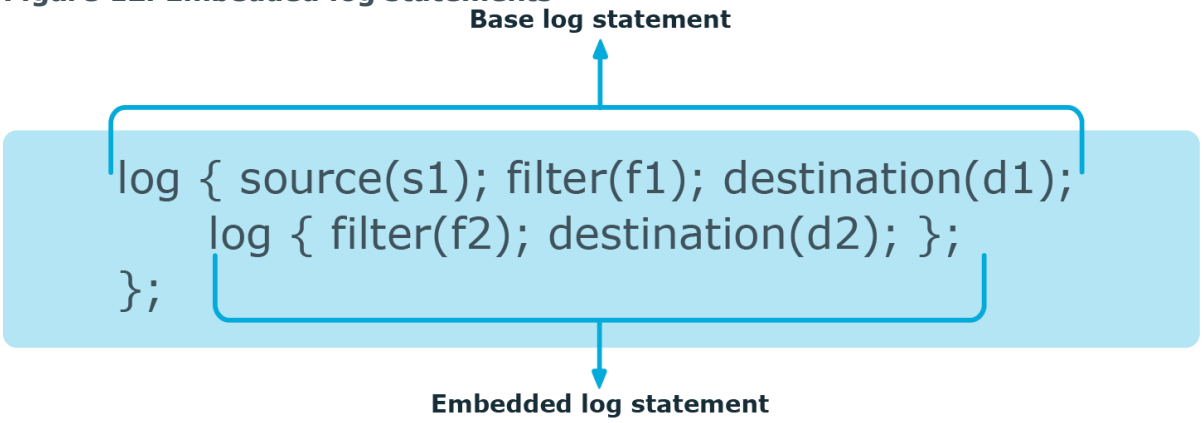
Figure 11: Embedded log statement



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 12: 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 to 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);
       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 OSE, 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 OSE 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 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 OSE 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 the 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

Flag	Description
	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 OSE 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 the 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 OSE treats the message as 'processed'. Since the first log path includes the `final` flag, syslog-ng OSE will not pass the message to the second log path (the one with the `fallback` flag). As a result, syslog-ng OSE 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 OSE 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 OSE would check if the message complies with the second `if` condition, that is, whether or not the message contains the string `bar` .)

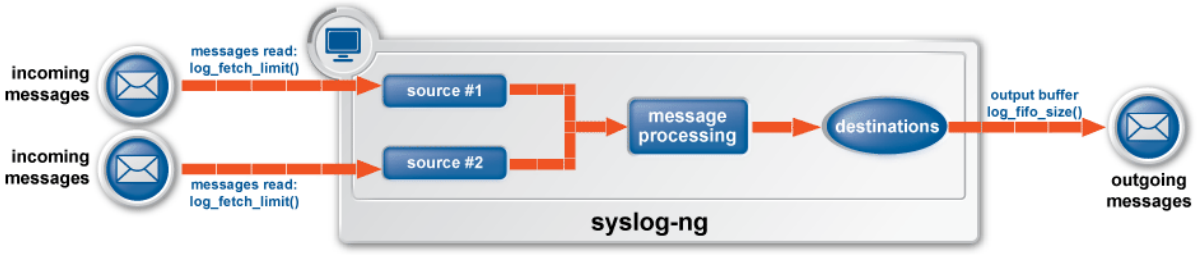
Managing incoming and outgoing messages with flow-control

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

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

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

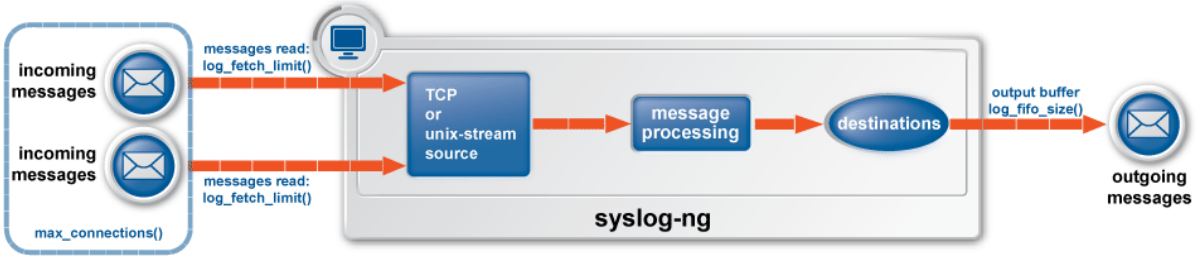
Figure 13: Managing log messages in syslog-ng



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

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

Figure 14: Managing log messages of TCP sources in syslog-ng



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

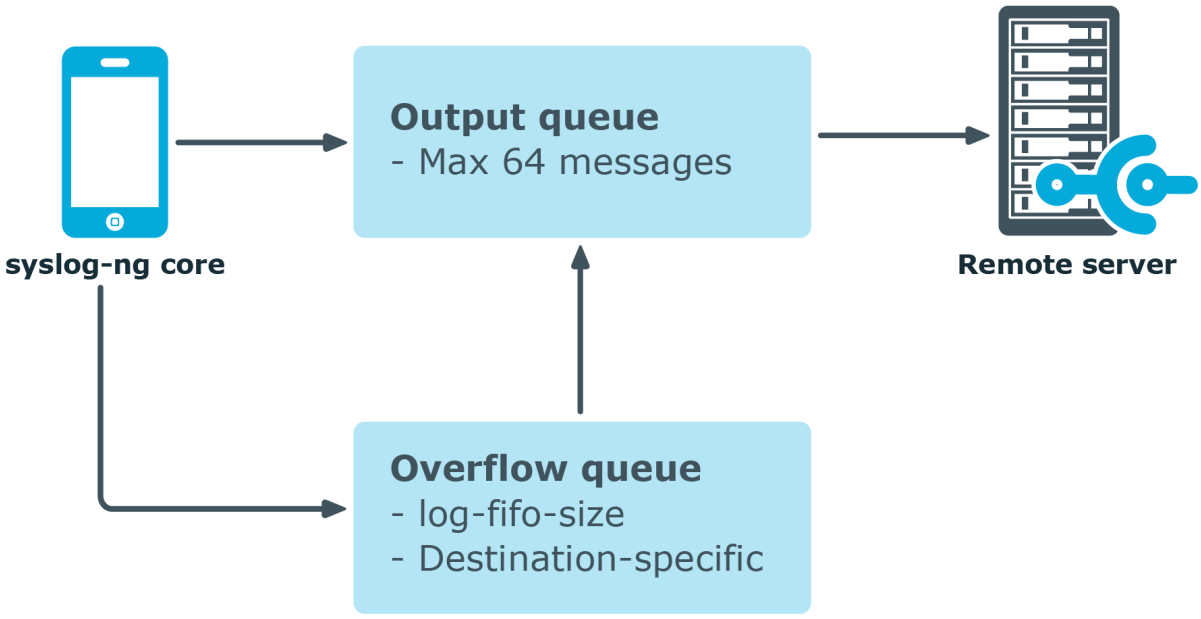
NOTE:

If the source can handle multiple connections (for example, `network()`), the size of the control window is divided by the value of the `max-connections()` parameter and this smaller control window is applied to each connection of the source.

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

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

Figure 15: Handling outgoing messages in syslog-ng OSE



- *Output queue*: Messages from the output queue are sent to the target syslog-ng server. The syslog-ng application puts the outgoing messages directly into the output queue, unless the output queue is full. The output queue can hold 64 messages, this is a fixed value and cannot be modified.
- *Disk buffer*: If the output queue is full and disk-buffering is enabled, syslog-ng puts the outgoing messages into the disk buffer of the destination.
- *Overflow queue*: If the output queue is full and the disk buffer is disabled or full, syslog-ng puts the outgoing messages into the overflow queue of the destination. (The overflow queue is identical to the output buffer used by other destinations.) The `log-fifo-size()` parameter specifies the number of messages stored in the overflow queue. For details on sizing the `log-fifo-size()` parameter, see 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 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, 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);
};
```

Flow-control and multiple destinations

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

NOTE:

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

If you use flow-control and reliable disk-based buffering together with multiple destinations, the flow-control starts slowing down the source only when:

- one destination is down, and
- the number of messages stored in the disk buffer of the destination reaches (`disk-buf-size()` minus `mem-buf-size()`).

Configuring flow-control

For details on how flow-control works, see 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 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. Therefore, set the `max-connections()` parameter of the source to `300`. However, the `log-fetch-limit()` (default value: 10) parameter applies to every connection of the source individually, while the `log-iw-size()` (default value: 1000) parameter applies to the source. In a worst-case scenario, the destination does not accept any messages, while all 300 connections send at least `log-fetch-limit()` number of messages to the source during every poll loop. Therefore, the control window must accommodate at least `max-connections()*log-fetch-limit()` messages to be able to read every incoming message of a poll loop. In the current example this means that (`log-iw-size()` should be greater than `300*10=3000`. If the control window is smaller than this value, the control window might fill up with messages from the first connections — causing syslog-ng to read only one message of the last connections in every poll loop.

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

```
source s_localhost {
    network(
        ip(127.0.0.1)
        port(1999)
        max-connections(300)
    );
};
destination d_tcp {
    network("10.1.2.3"
        port(1999)
        localport(999)
        log-fifo-size(30000)
    );
```

```
};
log {
    source(s_localhost);
    destination(d_tcp);
    flags(flow-control);
};
```

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

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

Using disk-based and memory buffering

The syslog-ng Open Source 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 OSE application automatically sends the stored messages to the destination when the connection is reestablished. The disk buffer is used as a queue: when the connection to the destination is reestablished, syslog-ng OSE sends the messages to the destination in the order they were received.

NOTE:

Disk-based buffering can be used in conjunction with flow-control. For details on flow-control, see Managing incoming and outgoing messages with flow-control.

Every such destination uses a separate disk buffer (similarly to the output buffers controlled by `log-fifo-size()`). The hard disk space is not pre-allocated, so ensure that there is always enough free space to store the disk buffers even when the disk buffers are full.

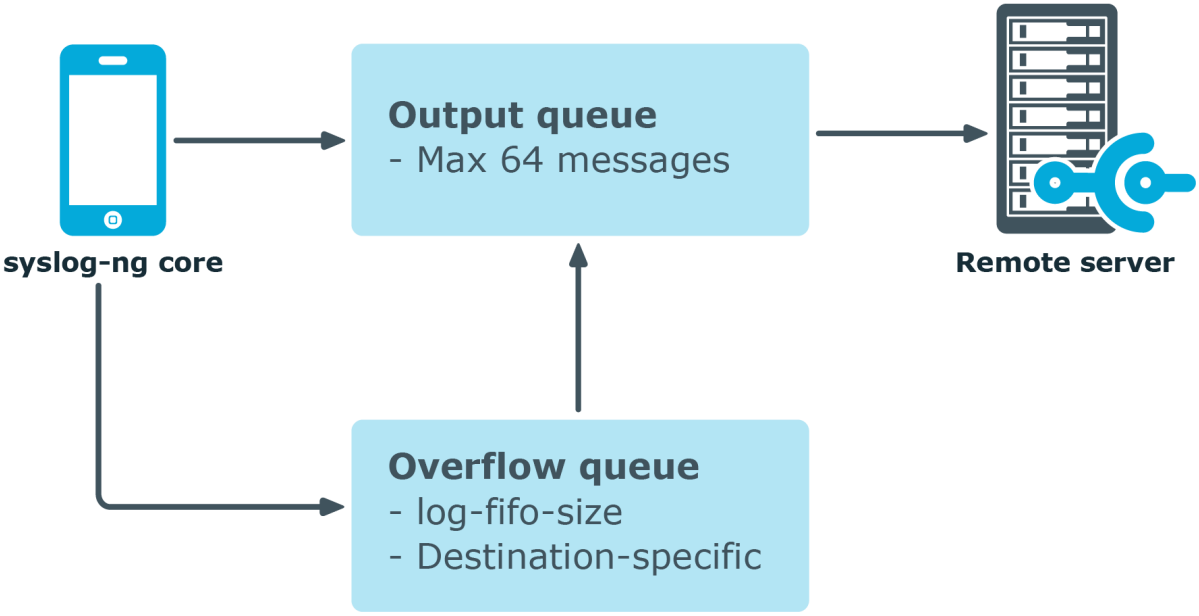
If syslog-ng OSE is restarted (using the `/etc/init.d/syslog-ng restart` command, or another appropriate command on your platform), it automatically saves any unsent messages from the disk buffer and the output queue. After the restart, syslog-ng OSE sends the saved messages to the destination. In other words, the disk buffer is persistent. The disk buffer is also resistant to syslog-ng OSE crashes.

The syslog-ng OSE application supports two types of disk buffering: reliable and normal. For details, see Enabling reliable disk-based buffering and Enabling normal disk-based buffering, respectively.

Message handling and normal disk-based buffering

When you use disk-based buffering, and the `reliable()` option is set to `no`, syslog-ng OSE handles outgoing messages the following way:

Figure 16: Handling outgoing messages in syslog-ng OSE



- *Output queue*: Messages from the output queue are sent to the destination (for example, your central log server). The syslog-ng OSE application puts the outgoing messages directly into the output queue, unless the output queue is full. By default, the output queue can hold 64 messages (you can adjust it using the `quot-size()`

option).

- *Disk buffer*: If the output queue is full, disk-buffering is enabled, and `reliable()` is set to `no`, syslog-ng OSE puts the outgoing messages into the disk buffer of the destination. (The disk buffer is enabled if the `disk-buffer()` option is configured.)
- *Overflow queue*: If the output queue is full and the disk buffer is disabled or full, syslog-ng OSE puts the outgoing messages into the overflow queue of the destination. (The overflow queue is identical to the output buffer used by other destinations.) The `log-fifo-size()` parameter specifies the number of messages stored in the overflow queue. For details on sizing the `log-fifo-size()` parameter, see also Managing incoming and outgoing messages with flow-control.

NOTE:

Using disk buffer can significantly decrease performance.

Message handling and reliable disk-based buffering

When you use disk-based buffering, and the `reliable()` option is set to `yes`, syslog-ng OSE handles outgoing messages the following way.

The `mem-buf-size()` option determines when flow-control is triggered. All messages arriving to the log path that includes the destination using the disk-buffer are written into the disk-buffer, until the size of the disk-buffer reaches (`disk-buf-size()` minus `mem-buf-size()`). Above that size, messages are written into both the disk-buffer and the memory-buffer, indicating that flow-control needs to slow down the message source. These messages are not taken out from the control window (governed by `log-iw-size()`), causing the control window to fill up. If the control window is full, the flow-control completely stops reading incoming messages from the source. (As a result, `mem-buf-size()` must be at least as large as `log-iw-size()`.)

Enabling reliable disk-based buffering

To enable reliable disk-based buffering, use the `disk-buffer(reliable(yes))` parameter in the destination. Use reliable disk-based buffering if you do not want to lose logs in case of reload/restart, unreachable destination or syslog-ng OSE 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 reliable disk-based buffering

```
destination d_BSD {
    network("127.0.0.1"
        port(3333)
```

```
        disk-buffer(
            mem-buf-size(10000)
            disk-buf-size(2000000)
            reliable(yes)
        )
    );
};
```

For details on the differences between normal and reliable disk-based buffering, see also About disk queue files.

Enabling normal disk-based buffering

To enable normal disk-based buffering, use the `disk-buffer(reliable(no))` parameter in the destination. Use normal disk-based buffering if you want a solution that is faster than the reliable disk-based buffering. In this case, disk buffering will be less reliable and it is possible to lose logs in case of syslog-ng OSE crash. The filename of the normal disk buffer file is the following: `<syslog-ng path>/var/syslog-ng-00000.qf`.

Example: Example for using normal disk-based buffering

When using the disk-buffer plugin:

```
destination  d_BSD  {
       network("127.0.0.1"
              port(3333)
              disk-buffer(
                     mem-buf-length(10000)
                     disk-buf-size(2000000)
                     reliable(no)
              )
       );
 };
```

For details on the differences between normal and reliable disk-based buffering, see also About disk queue files.

Enabling memory buffering

To enable memory buffering, use the `log-fifo-size()` parameter in the destination. All destination drivers can use memory buffering. Use memory buffering if you want to send

logs to destinations where disk-based buffering is not available. Or if you want the fastest solution, and if syslog-ng OSE crash or network downtime is never expected. In these cases, losing logs is possible. This solution does not use disk-based buffering, logs are stored only in the memory.

Example: 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 OSE crashes if the client and the destination server communicate using the Reliable Log Transfer Protocol (RLTP). Note that the Reliable Log Transfer Protocol is available only in syslog-ng Premium Edition. Of course, using the `reliable(yes)` option introduces a significant performance penalty as well. Reliable disk queues employ an in-memory cache buffer, the content of which is also written to the disk, and which is intended to speed up the process of reading back data from the queue.

Normal disk queues work in a different way: they employ an in-memory output buffer (set in `qout-size()`) and an in-memory overflow queue (set in `mem-buf-length()`). The disk buffer file itself is only used if the overflow buffer is filled up completely. This approach has better performance (because of less disk IO operations), but also carries the risk of losing a maximum of `qout-size()` plus `mem-buf-length()` number of messages in case of an unexpected power failure or application crash.

Size and truncation of 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. Truncation only occurs if the read and write heads of the queue reach the same position. Given that new messages arrive all the time, at least a small number of messages will almost always be stored in the queue file at all times. As a result, the queue file is not truncated automatically, but grows until it

reaches the maximal configured size, after which the write head will wrap around, later followed by the read head.

In case of normal disk queue files, growth in size is not so apparent, as the disk-based queue file is only used if the in-memory overflow buffer fills up. Once the destination sends messages faster than the incoming message rate, the queue will start to empty, and when the read and write heads of the queue reach the same position, the queue files are finally truncated.

Note that if a queue file becomes corrupt, syslog-ng OSE 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 OSE.

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 OSE version 3.2, 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 Customize 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 higher than 5.

```
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 12: 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 OSE 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 OSE, 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 OSE version 3.0 and 3.1. In syslog-ng OSE 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 OSE 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.

NOTE:

- Tagging messages and also filtering on the tags is very fast, much faster than other types of filters.
- Tags are available locally, that is, if you add tags to a message on the client, these tags will not be available on the server.
- To include the tags in the message, use the `${TAGS}` macro in a template. Alternatively, if you are using the IETF-syslog message format, you can include the `${TAGS}` macro in the `.SDATA.meta` part of the message. Note that the `${TAGS}` macro is available only in syslog-ng OSE 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 13: Filter functions available in syslog-ng OSE

Name	Description
facility()	Filter messages based on the sending facility.
filter()	Call another filter function.
host()	Filter messages based on the sending host.
inlist()	File-based whitelisting and blacklisting.
level() or priority()	Filter messages based on their priority.
match()	Use a regular expression to filter messages based on a specified header or content field.
message()	Use a regular expression to filter messages based on their content.
netmask() or 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 OSE 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 14: 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 OSE will compare the filter expression to the content of the ${HOST} macro. This means that for the IP address of a

host will not match, even if the IP address and the hostname field refers to the same host. To filter on IP addresses, use the `netmask()` filter.

```
filter demo_filter { host("example") };
```

inlist()

Synopsis: in-list("</path/to/file.list>", value("<field-to-filter>"))

Description: Matches the value of the specified field to a list stored in a file, allowing you to do simple, file-based black- and whitelisting. The file must be a plain-text file, containing one entry per line. The syslog-ng OSE 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 OSE for the changes to take effect.

Available in syslog-ng OSE 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 OSE), 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 OSE), 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 OSE will ignore the bits of the address after the prefix. To filter IPv4 addresses, see netmask().

The `netmask6()` filter is available in syslog-ng OSE 3.7 and later.

CAUTION:

If the IP address is not syntactically correct, the filter will never match. The syslog-ng OSE 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 OSE 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 OSE

Configuring global syslog-ng options
Global options

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 OSE 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 OSE 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 OSE server via a syslog-ng OSE relay, the hostname depends on the settings of the `keep-hostname()` and the `chain-hostnames()` options both on the syslog-ng OSE relay and the syslog-ng OSE 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-resolved-on-the-server	client-hostname-resolved-on-the-relay	

The `chain-hostnames()` option of syslog-ng can interfere with the way syslog-ng OSE 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.

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 3.16, the default value of this option is -1, so syslog-ng OSE 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 OSE 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 OSE 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 OSE or in case of network sources, the connection with the client is closed, syslog-ng OSE 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 application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

group()

Accepted values:	groupid
Default:	root

Description: The default group of output files. By default, syslog-ng changes the privileges of accessed files (for example `/dev/null`) to `root.root 0600`. To disable modifying privileges, use this option with the `-1` value.

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the syslog-ng OSE 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 OSE 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 OSE 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 OSE. For details on using name resolution in syslog-ng OSE, see Using name resolution in syslog-ng.
s

NOTE:

If the log message does not contain a hostname in its `HOST` field, syslog-ng OSE 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 OSE 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 OSE server and also on every relay, otherwise syslog-ng OSE 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 OSE).

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 maximal value that can be set is 268435456 bytes (256MB). For messages using the IETF-syslog message format (RFC5424), the maximal size of the value of an SDATA field is 64kB.

In most cases, it is not recommended 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.

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 OSE 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 OSE.

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 OSE 3.4 and later.

normalize-hostnames()

Accepted values:	yes | no
Default:	no

Description: If enabled (`normalize-hostnames(yes)`), syslog-ng OSE 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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `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 3.16, the default value of this option is -1, so syslog-ng OSE does not change the ownership, unless explicitly configured to do so.

pass-unix-credentials()

Accepted values:	yes|no
Default:	yes

Description: Enable syslog-ng OSE 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 Open Source 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:	0600

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 3.16, the default value of this option is -1, so syslog-ng OSE 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 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.

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.

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.

stats-freq()

Accepted values:	number
Default:	600

Description: The period between two STATS messages in seconds. STATS are log messages sent by syslog-ng, containing statistics about dropped log messages. Set to `0` to disable the STATS messages.

stats-level()

Accepted values:	`0 | 1 | 2 | 3`
Default:	`0`

Description: Specifies the detail of statistics syslog-ng collects about the processed messages.

- Level 0 collects only statistics about the sources and destinations.
- Level 1 contains details about the different connections and log files, but has a slight memory overhead.
- Level 2 contains detailed statistics based on the hostname.
- Level 3 contains detailed statistics based on various message parameters like facility, severity, or tags.

Note that level 2 and 3 increase the memory requirements and CPU load. For details on message statistics, see Statistics of syslog-ng.

stats-max-dynamics()

Accepted values:	number
Default:	N/A

Description: To avoid performance issues or even overloading syslog-ng OSE (for example, if a script starts to send logs from different IP addresses to syslog-ng OSE), you might want to limit the number of registered dynamic counters in the message statistics. For details on message statistics, see Statistics 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 OSE, 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 OSE to run in multithreaded mode and use multiple CPUs. Available only in syslog-ng Open Source Edition 3.3 and later. Note that setting `threaded(no)` does not mean that syslog-ng OSE will use only a single thread. For details, see Multithreading and scaling in syslog-ng OSE.

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
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.

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.

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 OSE application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all hosts which may get to syslog-ng are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Add Fully Qualified Domain Name instead of short hostname. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

NOTE:

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

use-rcptid()

Accepted values:	yes | no
Default:	no

Description: When the `use-rcptid` global option is set to `yes`, syslog-ng OSE 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).

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

TLS-encrypted message transfer

Secure logging using TLS
Encrypting log messages with TLS
Mutual authentication using TLS
Password-protected keys
TLS options

Secure logging using TLS

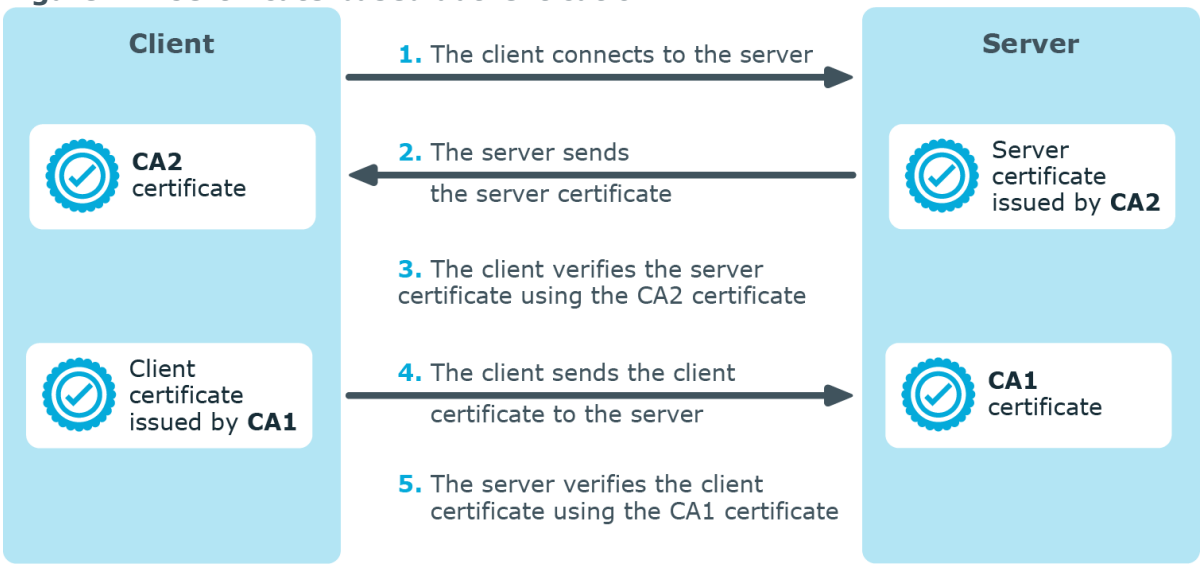
The syslog-ng application can send and receive log messages securely over the network using the Transport Layer Security (TLS) protocol using the `network()` and `syslog()` drivers.

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 17: 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.

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):

Steps:

1. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng server (or the self-signed certificate of the syslog-ng server) to the syslog-ng client hosts, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.

   Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem` The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

   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
   ```

2. Add a destination statement to the syslog-ng configuration file that uses the `tls( ca-dir(path_to_ca_directory) )` option and specify the directory using the CA certificate. The destination must use the `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"))
   ```

```
    );
};
```

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

CAUTION:

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

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

Configuring TLS on the syslog-ng server

Purpose:

Complete the following steps on the syslog-ng server:

Steps:

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

Purpose:

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

Steps:

1. Create an X.509 certificate for the syslog-ng client.
2. Copy the certificate (for example `client_cert.pem`) and the matching private key (for example `client.key`) to the syslog-ng client host, for example into the `/opt/syslog-ng/etc/syslog-ng/cert.d` directory. The certificate must be a valid X.509 certificate in PEM format. If you want to use a password-protected key, see Password-protected keys.
3. Copy the CA certificate of the Certificate Authority (for example `cacert.pem`) that issued the certificate of the syslog-ng server (or the self-signed certificate of the syslog-ng server) to the syslog-ng client hosts, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.

   Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem` The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

   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
   ```
4. Add a destination statement to the syslog-ng configuration file that uses the `tls( ca-dir(path_to_ca_directory) )` option and specify the directory using the CA certificate. The destination must use the `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")
             )
       );
 };
```

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

CAUTION:

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

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

Configuring TLS on the syslog-ng server

Purpose:

Complete the following steps on the syslog-ng server:

Steps:

1. Copy the certificate (for example `syslog-ng.cert`) of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/cert.d`

directory. The certificate must be a valid X.509 certificate in PEM format.

2. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng clients to the syslog-ng server, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.

   Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem` The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

   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 OSE version 3.14, you can use password-protected private keys in the `network()` and `syslog()` source and destination drivers.

Restrictions and limitations:

- *Hazard of data loss!* If you use password-protected keys, you must provide the passphrase of the password-protected keys every time syslog-ng OSE is restarted (syslog-ng OSE 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 OSE 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 such cases, always use disk-based buffering 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 OSE 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 OSE as a non-root user). If you use lots of password-protected private keys in your syslog-ng OSE 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 OSE 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 OSE 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 OSE configuration will be unaffected. You must provide the passphrase of the password-protected keys every time syslog-ng OSE 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 OSE 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:

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 OSE 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 OSE application uses this certificate to authenticate the syslog-ng OSE 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 OSE 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 OSE. 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 $1}' | xargs
echo -n | sed 's/ /:/g' | sed -e 's/:$//')\")"
```

Note that starting with version 3.10, when syslog-ng OSE receives TLS-encrypted connections, the order of ciphers set on the syslog-ng OSE server takes precedence over the client settings.

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`.

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 OSE supports only DH parameter files in the PEM format. If you do not set this parameter, syslog-ng OSE 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).

This option is only available when syslog-ng is compiled with OpenSSL version 1.0.2 or later. In the case of older versions, prime256v1 (NIST P-256) is used.

The following example curves work for all versions of OpenSSL that are equal to or later than version 1.0.2:

```
ecdh-curve-list("prime256v1:secp384r1")
```

key-file()

Accepted values:	Filename
Default:	none

Description: The name of a file that contains an unencrypted private key in PEM format, suitable as a TLS key. If properly configured, the syslog-ng OSE application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the syslog-ng OSE client on the destination server.

peer-verify()

Accepted values:	`optional-trusted` | `optional-untrusted` | `required-trusted` | `required-untrusted` | `yes` | `no`
Default:	`required-trusted`

Description: Verification method of the peer, the four possible values is a combination of two properties of validation:

- whether the peer is required to provide a certificate (required or optional prefix), and
- whether the certificate provided needs to be valid or not.

The following table summarizes the possible options and their results depending on the certificate of the peer.

		The remote peer has:		
		no certificate	**invalid certificate**	**valid certificate**
Local peer-verify() setting	*optional-untrusted*	TLS-encryption	TLS-encryption	TLS-encryption
	optional-trusted	TLS-encryption	rejected connection	TLS-encryption
	required-untrusted	rejected connection	TLS-encryption	TLS-encryption
	required-trusted	rejected connection	rejected connection	TLS-encryption

For untrusted certificates only the existence of the certificate is checked, but it does not have to be valid — syslog-ng accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

CAUTION:

When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, syslog-ng OSE will reject the connection.

Starting with syslog-ng OSE version 3.10, you can also use a simplified configuration method for the `peer-verify` option, simply setting it to `yes` or `no`. 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

pkcs12-file()

Accepted values:	Filename
Default:	none

Description: The name of a PKCS #12 file that contains an unencrypted private key, an X.509 certificate, and an optional set of trusted CA certificates.

If this option is used in the configuration, the value of `key-file()` and `cert-file()` will be omitted.

You can use the `ca-dir()` option together with `pkcs12-file()`. However, this is optional because the PKCS #12 file may contain CA certificates as well.

Passphrase is currently not supported.

Example: Using `pkcs12-file()`

In the following example, the first command creates a single PKCS #12 file from the private key, X.509 certificate, and CA certificate files. Then, the second half of the example uses the same PKCS #12 file in the syslog-ng configuration.

Example:

```
$ openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca.crt -
out server.p12
```

Example configuration:

```
source s_tls {
    syslog(
        transport(tls)
        tls(
            pkcs12-file("/path/to/server.p12")
            ca-dir("/path/to/cadir") # optional
```

```
            peer-verify(yes)
        )
    );
};
```

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 OSE 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 OSE 3.7 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.

13

template and rewrite: Format, modify, and manipulate log messages

This chapter explains the methods that you can use to customize, reformat, and modify log messages using syslog-ng Open Source Edition.

- Customize 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 OSE objects like filters and rewrite rules.

Customize 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 OSE lists the different types of macros available in syslog-ng OSE.
- Using template functions explains what template functions are and how to use them.
- Template functions of syslog-ng OSE lists the template functions available in syslog-ng OSE.

Formatting messages, filenames, directories, and tablenames

The syslog-ng OSE 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 OSE, see Macros of syslog-ng OSE.
- 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 OSE 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 Open Source Edition, see Macros of syslog-ng OSE. 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} ${MESSAGE}\n") );
};
```

Macros can be included by prefixing the macro name with a `$` sign, just like in Bourne compatible shells. Although using braces around macro names is not mandatory, and the `"$MESSAGE"` and `"${MESSAGE}"` formats are equivalent, using the `"${MESSAGE}"` format is recommended for clarity.

To use a literal `$` character in a template, you have to escape it. In syslog-ng OSE versions 3.4 and earlier, use a backslash (`\$`). In version 3.5 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} ${MSGHDR}${MESSAGE}\n`. (The `${MSGHDR}${MESSAGE}` 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} ${MESSAGE}\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} ${MESSAGE}\n");
};
template t_demo_template-without-braces "${ISODATE} ${HOST} ${MESSAGE}\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}  ${MESSAGE}\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 OSE).

- `R_` prefix, for example, `${R_DATE}`: `${R_DATE}` is the date when syslog-ng OSE has received the message.

- C_ prefix, for example, ${C_DATE}: ${C_DATE} is the current date, that is when syslog-ng OSE processes the message and resolves the macro.

The ${DATE} macro equals the ${S_DATE} macro.

The values of the date-related macros are calculated using the original timezone information of the message. To convert it to a different timezone, use the time-zone() option. You can set the time-zone() option as a global option, or per destination. For sources, it applies only if the original message does not contain timezone information. Converting the timezone changes the values of the following date-related macros (macros MSEC and USEC are not changed):

- AMPM
- DATE
- DAY
- FULLDATE
- HOUR
- HOUR12
- ISODATE
- MIN
- MONTH
- MONTH_ABBREV
- MONTH_NAME
- MONTH_WEEK
- SEC
- STAMP
- TZ
- TZOFFSET
- UNIXTIME
- WEEK
- WEEK_DAY
- WEEK_DAY_ABBREV
- WEEK_DAY_NAME
- YEAR
- YEAR_DAY

Hard vs. soft macros

Hard macros contain data that is directly derived from the log message, for example, the ${MONTH} macro derives its value from the timestamp. Hard macros are read-only. Soft macros (sometimes also called name-value pairs) are either built-in macros automatically

generated from the log message (for example, ${HOST}), or custom user-created macros generated by using the syslog-ng pattern database or a CSV-parser. In contrast to hard macros, soft macros are writable and can be modified within syslog-ng OSE, 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 OSE 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 OSE

The following macros are available in syslog-ng OSE.

CAUTION:

These macros are available when syslog-ng OSE 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 OSE 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 OSE 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 OSE received the message (only available if syslog-ng OSE 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 OSE (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 OSE, 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 OSE application uses the following procedure to determine the value of the `$FULLHOST_FROM` macro:

1. The syslog-ng OSE application takes the IP address of the host sending the message.
2. If the `use-dns()` option is enabled, syslog-ng OSE 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 OSE, 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 OSE 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 OSE (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 OSE, 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 OSE application uses the following procedure to determine the value of the $HOST_FROM macro:

1. The syslog-ng OSE application takes the IP address of the host sending the message.
2. If the use-dns() option is enabled, syslog-ng OSE 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 OSE, 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 can produce fractions of a second (for example milliseconds) in the timestamp by using the frac-digits() global or per-destination option.

LEVEL_NUM

Description: The priority (also called severity) of the message, represented as a numeric value, for example, 3. For the textual representation of this value, use the ${LEVEL} macro. See PRIORITY or LEVEL for details.

LOGHOST

Description: The hostname of the computer running syslog-ng OSE — 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 OSE 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 OSE 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 OSE version 3.4 and later.

MSG

The ${MSG} macro is an alias of the ${MESSAGE} macro, using ${MSG} in syslog-ng OSE 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 OSE 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 OSE 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 3.16 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 OSE 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 OSE 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.

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 OSE 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 OSE stores the sequence number from the message in this macro. If you forward this message the IETF-syslog protocol, syslog-ng OSE 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 OSE 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 OSE is reloaded, but it is reset when syslog-ng OSE 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 OSE receives, use the RCPTID macro.

SOURCE

Description: The identifier of the source statement in the syslog-ng OSE configuration file that received the message. For example, if syslog-ng OSE 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 OSE instance was started (that is, the uptime of the syslog-ng OSE process). The value of this macro is an integer containing the time in 1/100th of the second.

Available in syslog-ng OSE version 3.4 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.

> 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`.

.TLS.X509

Description: When using a transport that uses TLS, these macros contain information about the peer's certificate. That way, you can use information from the client certificate in filenames, database values, or as other metadata. If you clients have their own certificates, then these values are unique per client, but unchangeable by the client. The following macros are available in syslog-ng OSE version 3.9 and later.

- `.TLS.X509_CN`: The Common Name of the certificate.
- `.TLS.X509_O`: The value of the Organization field.
- `.TLS.X509_OU`: The value of the Organization Unit field.

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 OSE version 3.7 and later.

USEC, C_USEC, R_USEC, S_USEC

Description: The microsecond the message was sent.

Available in syslog-ng OSE version 3.4 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 OSE. Template functions use the following syntax:

```
$(function-name parameter1 parameter2 parameter3 ...)
```

For example, the `$(echo)` template function simply returns the value of the macro it receives as a parameter, thus `$(echo ${HOST})` is equivalent to `${HOST}`.

The parameters of template functions are separated by a whitespace character. If you want to use a longer string or multiple macros as a single parameter, enclose the parameter in double-quotes or apostrophes. For example:

```
$(echo "${HOST} ${PROGRAM} ${PID}")
```

Template functions can be nested into each other, so the parameter of a template function can be another template function, like:

```
$(echo $(echo ${HOST}))
```

For details on the available template functions, see the descriptions of the individual template functions in Template functions of syslog-ng OSE.

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}") ${MESSAGE}";
destination d_file {
      file("/tmp/mylogs.log" template("$(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=$(length "${MESSAGE}") ${MESSAGE}";
```

Template functions of syslog-ng OSE

The following template functions are available in syslog-ng OSE.

basename

Syntax:

```
$(basename argument)
```

Description: Returns the filename from an argument (for example, a macro: `$(basename ${FILE_NAME})`) that contains a filename with a path. For example, `$(basename "/var/log/messages.log")` returns `messages.log`. To extract the path, use the `dirname` template function.

Available in syslog-ng OSE version 3.10 and later.

context-lookup

Syntax:

```
$(context-lookup [option] condition value-to-select)
```

Description: The `context-lookup` 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 messages (for example, the ${MESSAGE}) as a list. It works similarly to the `$(grep)` template function, but it escapes its output properly, so that the returned value is a list that can be processed with other template functions that work on lists, for example, `$(list-slice)`.

Example: Using the `context-lookup` 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 `tags` name-value pair.

```
$(context-lookup ("${username}" == "root") ${tags})
```

To limit the number of matches that the template function returns, use the `--max-count` option, for example, `$(context-lookup --max-count 5 ("${username}" == "root") ${tags})`. If you do not want to limit the number of matches, use `--max-count 0`.

You can to specify multiple name-value pairs as parameters, separated with commas. If multiple messages match the condition of `context-lookup`, these will be returned also separated by commas. This can be used for example to collect the e-mail recipients from postfix messages.

Available in syslog-ng OSE version 3.10 and later.

context-values

Syntax:

```
$(context-values $name-value1 $name-value2 ...)
```

Description: The `context-values` template function returns a list of every occurrence of the specified name-value pairs from the entire context. For example, if the context contains

multiple messages, the `$(context-values ${HOST})` template function will return a comma-separated list of the `${HOST}` values that appear in the context.

Available in syslog-ng OSE version 3.10 and later.

dirname

Syntax:

```
$(dirname argument)
```

Description: Returns the path (without the filename) from an argument (for example, a macro: `$(basename ${FILE_NAME})` that contains a filename with a path. For example, `$(dirname "/var/log/messages.log")` returns `/var/log` path. To extract the filename, use the `basename` template function.

Available in syslog-ng OSE version 3.10 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 OSE 3.5 and later.

format-cef-extension

syslog-ng OSE version 3.8 includes a new template function (`format-cef-extension`) to format name-value pairs as ArcSight Common Event Format extensions. Note that the template function only formats the selected name-value pairs, it does not provide any mapping. There is no special support for creating the prefix part of a Common Event Format (CEF) message. Note that the order of the elements is random. For details on the CEF extension escaping rules format, see the ArcSight Common Event Format.

You can use the value-pairs that syslog-ng OSE 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 OSE 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 OSE GitHub repository.

> NOTE:
>
> To use the `format-cim()` template function, syslog-ng OSE must be compiled with JSON support. For details, see Compiling options of syslog-ng OSE. To see if your syslog-ng OSE 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 3.16 and later.

The follwing 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-gelf

Syntax:

```
$(format-gelf)
```

Description: Available in syslog-ng OSE 3.13 and later.

You can use the Graylog Extended Log Format (GELF) template together with the `graylog2()` destination to send syslog messages to Graylog. GELF is the native data format of Graylog.

Example: Using the format-gelf template function

The following configuration example shows how you can use the `format-gelf` template:

```
destination graylog2 {
       network(
              "127.0.0.1"
              port(12201)
              transport(tcp)
              template("$(format-gelf)")
       );
};
```

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 OSE 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 OSE 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 the case of syslog-ng macros starting with a dot (for example, `".SDATA.meta.sequenceID"`), `format-json` replaces the dot with an underscore character (for example, `{"_SDATA":{"meta":{"sequenceId":"55555"}}}`).

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")
       );
 };
```

geoip (DEPRECATED)

This template function is deprecated. Use geoip2 instead.

Syntax:

```
$(geoip <IPv4-address>)
```

Description: This template function returns the 2-letter country code of any IPv4 address or host. IPv6 addresses are not supported. Currently only the 2-letter codes are supported, and only from the default database. For example, `$(geoip $HOST)`

NOTE:

This template function is available only if syslog-ng OSE has been compiled with the `--enable-geoip` compiling option.

To retrieve additional GeoIP information, see Looking up GeoIP data from IP addresses (DEPRECATED).

geoip2

Syntax:

```
$(geoip2 --database <path-to-geoip2-database-file>
    [ --field "registered_country.names.ru" ] ${HOST})
```

Description: This template function exctracts specific fields from the mmdb database using the `--field` parameter. If you omit this parameter, the it returns the 2-letter country code of any IPv4/IPv6 address or host.

NOTE:

This template function is available only if syslog-ng OSE 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.

getent

Syntax:

```
$(getent)
```

Description: Available in syslog-ng OSE 3.13 and later.

You can use the `getent` template function to look up entries from the Name Service Switch libraries, such as, passwd, services, or protocols.

The following databases are supported:

- *passwd*

  Use this database to query data related to a user. Specify the user by either username or user ID. You can query the following data: username, user ID, group ID, GECOS field, home directory, or user shell.

  ```
  $(getent passwd testuser name)
  $(getent passwd testuser uid)
  $(getent passwd testuser gid)
  $(getent passwd testuser gecos)
  $(getent passwd testuser dir)
  $(getent passwd testuser shell)
  ```

  or

  ```
  $(getent passwd 1000 name)
  $(getent passwd 1000 uid)
  $(getent passwd 1000 gid)
  $(getent passwd 1000 gecos)
  $(getent passwd 1000 dir)
  $(getent passwd 1000 shell)
  ```

  The queried data is optional. When you do not query any data, the default behavior applies, which is as follows: user ID is returned for username, or username is returned for user ID.

  - Username `$(getent passwd testuser)` returns user ID `1000`.
  - User ID `$(getent passwd 1000)` returns username `testuser`.

- *group*

  Use this database to query group-related data. The group can be specified using either group ID or group name. You can query the following data: group name, group ID, and members.

  ```
  $(getent group adm name)
  $(getent group adm gid)
  $(getent group adm members)
  ```

The queried data is optional. The default behavior is as follows: group ID is returned for group name, or group name is returned for user ID.

  - Group name `$(getent group adm)` returns group ID `4`.
  - Group ID `$(getent group 4)` returns group name `adm`.

- *protocols*

  Use this database to translate protocol name to protocol ID, or protocol ID to protocol string.

  ```
  $(getent protocols tcp)
  $(getent protocols 6)
  ```

- *services*

  Use this database to translate service name to service ID, or service ID to service name.

  ```
  $(getent services http)
  $(getent services 80)
  ```

graphite-output

Syntax:

```
$(graphite-output parameters)
```

Description: Available in syslog-ng OSE 3.6 and later (Originally appeared in the syslog-ng OSE 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 OSE 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 e-mail recipients from postfix messages.

hash

Syntax:

```
$(<method> [opts] $arg1 $arg2 $arg3...)
```

Options:

`--length N, -l N`
Truncate the hash to the first N characters.

Description: Calculates a hash of the string or macro received as argument using the specified hashing method. If you specify multiple arguments, effectively you receive the hash of the first argument salted with the subsequent arguments.

`<method>` can be one of md5, md4, sha1, sha256, sha512 and "hash", which is equivalent to 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 OSE has been compiled with the `--enable-ssl` compile option and the `tfhash` module has been loaded.

By default, syslog-ng OSE 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 OSE 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 OSE. 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 OSE version 3.10, 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 OSE 3.5 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 OSE 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 OSE embeds a Python interpreter to process the messages. Note the following points:

- Currently only Python 2.7 is supported.
- The Python block must be a top-level block in the syslog-ng OSE configuration file.
- The Python block can contain multiple Python functions.
- 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 function, use the dot-notation. 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.
- 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.

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: 3.17

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: 3.17

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 OSE 3.5 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.

stardate

Syntax:

```
$(stardate [option] "<date-in-unixtime>")
```

Description: Converts a date in UNIXTIME (for example, ${UNIXTIME}) into stardate, displaying the year and the progress of the year in a number of digits (`YYYY.NNN`). You can set the number of digits using the `--digits` option, for example:

```
$(stardate --digits 2 "${R_UNIXTIME}")
```

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 OSE 3.5 and later.

urlencode

Syntax:

```
$(urlencode ${MSG} )\n")
```

Description: You can use the `urlencode` template together with the `telegram()` destination to send syslog messages to Telegram. The `urlencode` 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 OSE version 3.16 and later.

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 OSE 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 OSE 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 OSE. A dedicated Redis destination is available in syslog-ng OSE version 3.5. For details, see redis: 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 OSE 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 vs. soft macros). You can also rewrite the structured-data fields of messages complying to the RFC5424 (IETF-syslog) message format. Substitution rules use the following syntax:

Declaration:

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

The `type()` and `flags()` options are optional. The `type()` specifies the type of regular expression to use, while the `flags()` are the flags of the regular expressions. For details on regular expressions, see 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 db-parser: Process message content with a pattern database (patterndb)). Hard macros cannot be modified. For details on the hard and soft macros, see Hard vs. soft macros). Note that the rewrite operation completely replaces any previous value of that field. Use the following syntax:

Declaration:

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

Example: 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");
};
```

frac-digits()

Type:	number
Default:	0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

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.

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 OSE cannot convert some data to the specified type. By default, syslog-ng OSE 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 OSE.
- `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.

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.

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.

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.

Unsetting message fields

You can unset macros or fields of the message, including any user-defined macros created using parsers (for details, see parser: Parse and segment structured messages and db-parser: Process message content with a pattern database (patterndb)). Hard macros cannot be modified. For details on hard and soft macros, see Hard vs. 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 &fullvendor;, the developer of &abbrev;.)

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"));
};
```

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. (If you want to modify the names of message fields, see map-value-pairs: Rename value-pairs to normalize logs.)

- 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.*"))
  ```

`map-value-pairs`: Rename value-pairs to normalize logs

The `map-value-pairs()` parser allows you to map existing name-value pairs to a different set of name-value pairs. You can rename them in bulk, making it easy to use for log normalization tasks (for example, when you parse information from different log messages, and want to convert them into a uniform naming scheme). You can use the normal value-pairs expressions, similarly to value-pairs based destinations.

Available in syslog-ng OSE version 3.10 and later.

Declaration:

```
parser parser_name {
    map-value-pairs(
        <list-of-value-pairs-options>
```

```
    );
};
```

Example: Map name-value pairs

The following example creates a new name-value pair called `username`, adds the hashed value of the `.apache.username` to this new name-value pair, then adds the `webserver` prefix to the name of every name-value pair of the message that starts with `.apache`

```
parser  p_remap_name_values  {
       map-value-pairs(
             pair("username",  "'($sha1  $.apache.username)")
             key('.apache.*'  rekey(add-prefix("webserver")))
       );
 };
```

Conditional rewrites

Starting with 3.2, 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 OSE configuration file, as you do not need to create separate log paths to modify certain messages.

How conditional rewriting works

Purpose:

The following procedure summarizes how conditional rewrite rules (rewrite rules that have the `condition()` parameter set) work. The following configuration snippet is used to illustrate the procedure:

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

```
        condition(program("myapplication"))
    );
};
log {
    source(s1);
    rewrite(r_rewrite_set);
    destination(d1);
};
```

Steps:

1. The log path receives a message from the source (`s1`).
2. The rewrite rule (`r_rewrite_set`) evaluates the condition. If the message matches the condition (the PROGRAM field of the message is "myapplication"), syslog-ng OSE 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)));};
```

Adding and deleting tags

To add or delete a tag, you can use rewrite rules. To add a tag, use the following syntax:

```
rewrite  <name_of_the_rule>  {
       set-tag("<tag-to-add>");
 };
```

To delete a tag, use the following syntax:

```
rewrite  <name_of_the_rule>  {
       clear-tag("<tag-to-delete>");
 };
```

You cannot use macros in the tags.

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 OSE 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.

Declaration:

```
@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 a 16-character-long hash. This hash is generated by calculating the SHA-1 hash of the credit card number, selecting the first 64 bits of this hash, and representing this 64 bits in 16 characters.

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 OSE 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 OSE 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). 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

NOTE:

- The wildcards can match the / character.
- You cannot use the * and ? literally in the pattern.

Optimizing regular expressions

The `host()`, `match()`, and `program()` filter functions and some other syslog-ng objects accept regular expressions as parameters. But evaluating general regular expressions puts a high load on the CPU, which can cause problems when the message traffic is very high. Often the regular expression can be replaced with simple filter functions and logical operators. Using simple filters and logical operators, the same effect can be achieved at a much lower CPU load.

Example: 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");
};
```

14

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 OSE application provides the following possibilities to parse the messages, or parts of the messages:

- By default, syslog-ng OSE 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 The JSON parser The JSON parser.
- To parse XML-formatted messages, see The XML parser.
- To parse a specially-formatted date or timestamp, see Parsing dates and timestamps.
- To write a custom parser in Python, see The Python Parser.
- To identify and parse the messages using a pattern database, see db-parser: Process message content with a pattern database (patterndb).

The syslog-ng OSE 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 The Cisco Parser.
- Linux Audit (`auditd`) logs. For details, see The Linux Audit Parser.
- osquery result logs. For details, see osquery: Collect and parse osquery result logs.

- SNMP traps of the Net-SNMP's `snmptrapd` application. For details, see snmptrap: Read Net-SNMP traps.

Parsing syslog messages

By default, syslog-ng OSE 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 OSE, 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 OSE 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, kernel, no-hostname, no-multi-line, no-parse, sanitize-utf8, store-legacy-msghdr, syslog-protocol, validate-utf8
Default:	empty set

Description: Specifies the log parsing options of the source.

- *assume-utf8*: The `assume-utf8` flag assumes that the incoming messages are UTF-8 encoded, but does not verify the encoding. If you explicitly want to validate the UTF-8 encoding of the incoming message, use the `validate-utf8` flag.
- *empty-lines*: Use the `empty-lines` flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *expect-hostname*: If the `expect-hostname` flag is enabled, syslog-ng OSE will assume that the log message contains a hostname and parse the message accordingly. This is the default behavior for TCP sources. Note that pipe sources use the `no-hostname` flag by default.
- *kernel*: The `kernel` flag makes the source default to the `LOG_KERN | LOG_NOTICE` priority if not specified otherwise.

- *no-hostname*: Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way, syslog-ng OSE 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 OSE 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 OSE 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 ${MESSAGE} part of a syslog message. In this case, syslog-ng OSE generates a new syslog header (timestamp, host, and so on) automatically. Note that since `flags(no-parse)` disables message parsing, it interferes with other flags, for example, disables `flags(no-multi-line)`.
- *dont-store-legacy-msghdr*: By default, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.
- *sanitize-utf8*: When using the `sanitize-utf8` flag, syslog-ng OSE converts non-UTF-8 input to an escaped form, which is valid UTF-8.
- *store-raw-message*: Save the original message as received from the client in the `${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 3.16 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 OSE application can separate parts of log messages (that is, the contents of the ${MESSAGE} 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 OSE 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.

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

Names starting with a dot (for example, `.example`) are reserved for use by syslog-ng OSE. 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 vs. 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.MIN", "APACHE.TIMESTAMP.ZONE")
              delimiters("/: ")
              flags(escape-none)
              template("${APACHE.TIMESTAMP}")
       );
};
log {
       source(s_local);
       parser(p_apache);
       parser(p_apache_timestamp);
       destination(d_file);
};
```

Further examples:

- For an example on using the greedy option, see Example: Adding the end of the message to the last column.

Options of CSV parsers

The syslog-ng OSE application can separate parts of log messages (that is, the contents of the ${MESSAGE} 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 OSE 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 OSE. 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 vs. 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 OSE 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>"))`

Multiple delimiters:

If you use more than one delimiter, note the following points:

- syslog-ng OSE 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 OSE 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-parsercsv-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.

null()

Synopsis: string

Description: If the value of a column is the value of the `null()` parameter, syslog-ng OSE changes the value of the column to an empty string. For example, if the columns of the message contain the "N/A" string to represent empty values, you can use the `null("N/A")` option to change these values to empty stings.

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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

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.

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 OSE 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 OSE 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 OSE 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 OSE stores only the last (rightmost) value for the key. Using the previous example, syslog-ng OSE will store the following pairs: `key1=value5, key2=value2, key3=value4`.

CAUTION:

If the names of keys in the message is the same as the names of syslog-ng OSE 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 vs. soft macros.

The parser discards message sections that are not `key=value` pairs, even if they appear between `key=value` pairs that can be parsed.

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 OSE 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 OSE 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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

For example, to insert the `postfix` prefix when parsing Postfix log messages, use the `prefix(.postfix.)` option.

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(":"));`

The JSON parser

The JSON parser

JavaScript Object Notation (JSON) is a text-based open standard designed for human-readable data interchange. It is used primarily to transmit data between a server and web application, serving as an alternative to XML. It is described in RFC 4627. The syslog-ng OSE application can separate parts of incoming JSON-encoded log messages to name-value pairs. For details on using value-pairs in syslog-ng OSE 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 OSE 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 OSE 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 vs. soft macros.

NOTE:

The JSON parser currently supports only integer, double and string values when interpreting JSON structures. As syslog-ng does not handle different data types internally, the JSON parser converts all JSON data to string values. In case of boolean types, the value is converted to 'TRUE' or 'FALSE' as their string representation.

The JSON parser discards messages if it cannot parse them as JSON messages, so it acts as a JSON-filter as well.

To create a JSON parser, define a parser that has the `json-parser()` option. Defining the prefix and the marker are optional. By default, the parser will process the `${MESSAGE}` part of the log message. To process other parts of a log message with the JSON parser, use the `template()` option. You can also define the parser inline in the log path.

Declaration:

```
parser  parser_name  {
       json-parser(
             marker()
             prefix()
       );
};
```

Example: 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 OSE 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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

template()

Synopsis: template("${<macroname>}")

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

The 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 simply concatenates the different parts of text. For example, from this input XML:

```
<tag>
  <tag1>text1</tag1>
  <tag1>text2</tag1>
 </tag>
```

The following output is generated:

```
.xml.tag.tag1 = text1text2
```

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.

Limitations

The XML parser comes with certain limitations.

Vector-like structures:

It is not possible to address each element of a vector-like structure individually. For example, take this input:

```
<vector>
       <entry>value1</entry>
       <entry>value2</entry>
       ...
       <entry>valueN</entry>
 </vector>
```

After parsing, the entries cannot be addressed individually. Instead, the text of the entries will be concatenated:

```
vector.entry = "value1value2...valueN"
```

Note that `xmllint` has the same behavior:

```
$ xmllint --xpath "/vector/entry/text()" test.xml
value1value2valueN%
```

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 XML parsers

The XML parser has the following options.

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</innertag></t
ag3>
```

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 OSE. 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 vs. 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 ${MESSAGE} 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 `time-stamp(recvd)` option.)

Note that parsing will fail if the format string does not match the entire template or field. Since by default syslog-ng OSE 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 OSE 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 OSE 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);
    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 ${MESSAGE} 0 24)") );
```

Options of `date-parser()` parsers

The `date-parser()` parser has the following options.

format()

Synopsis:	format(string)
Default:	

Description: Specifies the format how syslog-ng OSE 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)
%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(stamp)`, syslog-ng OSE adds the parsed date to the S_ macros (corresponding to the sent date). If you use `time-stamp(recvd)`, syslog-ng OSE adds the parsed date to the R_ macros (corresponding to the received date).

time-zone()

Synopsis:	time-zone(string)
Default:	

Description: If this option is set, syslog-ng OSE 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 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.

The Apache Access Log Parser

The Apache Access Log Parser can parse the access log messages of the Apache HTTP Server. The syslog-ng OSE application can separate these log messages to name-value pairs. For details on using value-pairs in syslog-ng OSE see Structuring macros, metadata, and other value-pairs. The `apache-accesslog-parser()` supports both the Common Log Format and the Combined Log Format of Apache (for details, see the Apache HTTP Server documentation). The following is a sample log message:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

The syslog-ng OSE application extracts every field into name-value pairs, and adds the `.apache.` prefix to the name of the field.

Declaration:

```
parser  parser_name  {
       apache-accesslog-parser(
             prefix()
       );
 };
```

The parser extracts the following fields from the messages: `clientip`, `ident`, `auth`, `timestamp`, `rawrequest`, `response`, `bytes`, `referrer`, and `agent`. The `rawrequest` field is further segmented into the `verb`, `request`, and `httpversion` fields. The syslog-ng OSE `apache-accesslog-parser()` parser uses the same naming convention as Logstash.

Example: Using the `apache-accesslog-parser` parser

In the following example, the source is a log file created by an Apache web server. The parser automatically inserts "`.apache.`" 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 statement connects the source, the destination, and the parser.

```
source  s_apache  {
       file(/var/log/access_log);
};

destination  d_json  {
       file(
              "/tmp/test.json"
              template("$(format-json  .apache.*)\n")
       );
};

log  {
       source(s_apache);
       parser  {  apache-accesslog-parser();};
       destination(d_json);
};
```

To use this parser, the `scl.conf` file must be included in your syslog-ng OSE configuration:

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

The `apache-accesslog-parser()` is actually a reusable configuration snippet configured parse Apache access log 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.

Options of `apache-accesslog-parser()` parsers

The `apache-accesslog-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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `apache-accesslog-parser()` uses the `.apache.` prefix. To modify it, use the following format:

```
parser  {
       apache-accesslog-parser(prefix("apache."));
 };
```

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}`).

The 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 OSE 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 OSE 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 OSE 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: 3.17
@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 OSE. 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 vs. 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."));
 };
```

The Linux Audit Parser

The Linux Audit Parser can parse the log messages of the Linux Audit subsystem (`auditd`). The syslog-ng OSE application can separate these log messages to name-value pairs. For details on using value-pairs in syslog-ng OSE see Structuring macros, metadata, and other value-pairs. 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 &abbrev; application automatically decodes these fields (for example, the `c000003e` value becomes `x86_64`).

The syslog-ng OSE 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 OSE 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.

Names starting with a dot (for example, `.example`) are reserved for use by syslog-ng OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

template()

Synopsis: template("${<macroname>}")

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

The Python Parser

The Python Log Parser (available in syslog-ng OSE version 3.10 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. For example, you can import external Python modules to process the messages, query databases to enrich the messages with additional data, and many other things.

- Available in syslog-ng OSE version 3.10 and later.
- Currently only Python 2.7 is supported.
- The Python block must be a top-level block in the syslog-ng OSE configuration file. If you store the Python code in a separate Python file and only include it in the syslog-ng OSE 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: `export PYTHONPATH=/opt/syslog-ng/etc`
- The Python object is initiated only once, when syslog-ng OSE is started or reloaded. That means it keeps the state of internal variables while syslog-ng OSE is running.
- The Python block can contain multiple Python functions.
- Using Python code in syslog-ng OSE can significantly decrease the performance of syslog-ng OSE, especially if the Python code is slow.
- Validate and lint the Python code before using it. The syslog-ng OSE application does not do any of this.

Declaration:

Python parsers consist of two parts. The first is a syslog-ng OSE parser object that you use in your syslog-ng OSE 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 {
import re
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.'''
        return True
    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 OSE application initializes Python objects only when it is started or reloaded. That means it keeps the state of internal variables while syslog-ng OSE 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 OSE 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 OSE will not start.

The return value of the `parse()` method must be `True`. If it returns `False`, or raises an exception, syslog-ng OSE 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 OSE configuration file (for example, in a template), use the `${new-macro-name}` macro.
- You cannot override hard macros (see Hard vs. 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 OSE is stopped or reloaded.

CAUTION:

It is common practice for log rotate solutions to reload syslog-ng OSE (by sending a HUP signal or using the operating system's init subsystem) and for users to execute `syslog-ng-ctl reload` (to start a configuration file reload). Care should be taken in these cases, because the methods and attributes defined in a Python parser block definition lose their context and state during a syslog-ng OSE 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
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
```

The syslog-ng OSE 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 OSE log statement uses the extracted fields to format the output message.

```
@version: 3.17
@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):
              return True
       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 OSE could process about 25000 real-life Windows log messages per second.

```
@version: 3.17
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):
        return True
    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 3.16 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();
 };
```

The sudo parser

The sudo parser can parse the log messages of the sudo command. Available in version 3.16 and later.

Declaration:

```
@version: [[[Undefined variable Version.configversion]]]
@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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`
(missing or bad snippet)

By default, `sudo-parser()` uses the `sudo.` prefix. To modify it, use the following format:

```
parser { sudo-parser(prefix("myprefix.")); };
```

The iptables parser

The iptables parser can parse the log messages of the iptables command. Available in version 3.16 and later.

Declaration:

```
@version: [[[Undefined variable Version.configversion]]]
@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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`
(missing or bad snippet)

By default, `iptables-parser()` uses the `iptables.` prefix. To modify it, use the following format:

```
parser { iptables-parser(prefix("myprefix.")); };
```

15

db-parser: Process message content with a pattern database (patterndb)

Classifying log messages
Using pattern databases
Triggering actions for identified messages
Creating pattern databases

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))?`

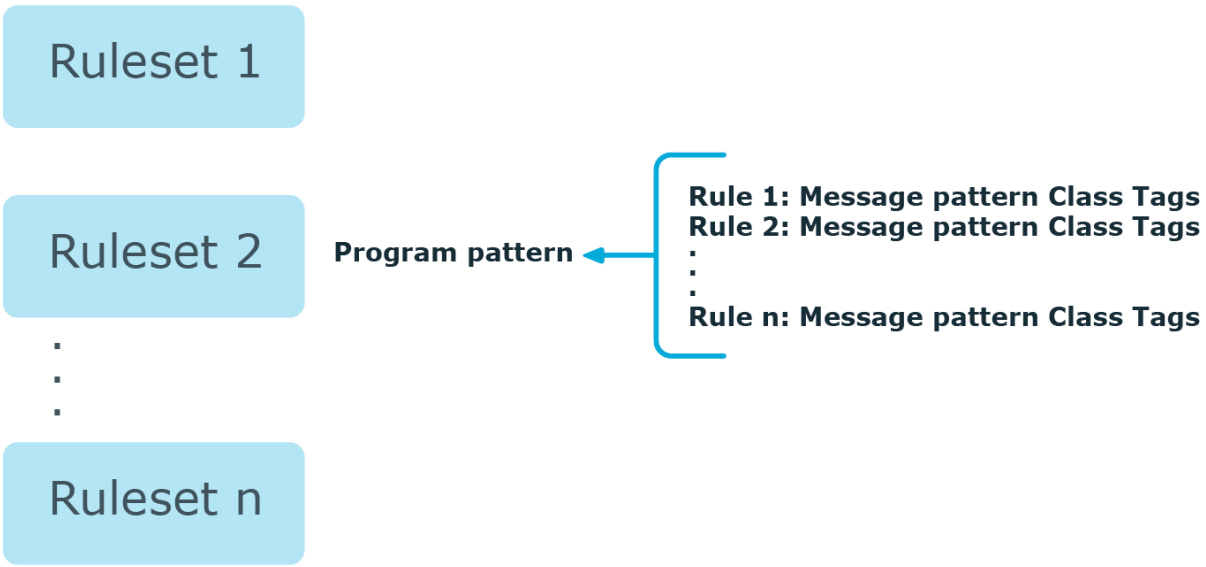
A syslog-ng database pattern for this message: `Accepted @QSTRING:auth_method: @ for@QSTRING:username: @from @QSTRING:client_addr: @port @NUMBER:port:@ ssh2`

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 18: 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 Open Source Edition in the e-mail 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.

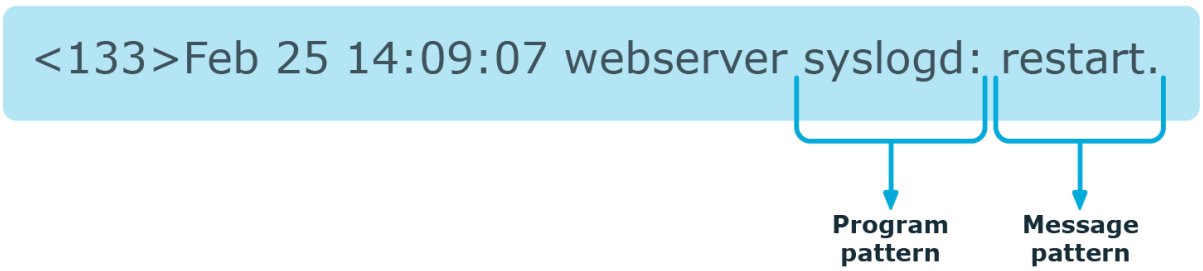
  NOTE:

  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 19: Applying patterns
A sample log message:



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

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

NOTE:

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

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

For example, if the message begins with the `Apple` string, only patterns beginning with the character `A` are considered. In the next step, syslog-ng selects the patterns that start with `Ap`, and so on, until there is no more specific pattern left. 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 to detect anomalies. When applied to log analysis, it means that you ignore the regular, common log messages - these are the result of the regular behavior of your system, and therefore are not too interesting. However, new messages that have not appeared in the logs before can sign important events, and should be therefore investigated. "By definition, something we have never seen before is anomalous" (Marcus J. Ranum).

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

you can iteratively classify more and more messages, until only the really anomalous messages show up as unknown.

Obviously, for this to work, a large number of message patterns are required. The radix-tree matching method used for message classification is very effective, can be performed very fast, and scales very well. Basically the time required to perform a pattern matching is independent from the number of patterns in the database. For sample pattern databases, see 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};${PID};${MESSAGE}\n")
             template-escape(no)
       );
 };
```

For details on how to create your own pattern databases see The syslog-ng pattern database format.

Drop unmatched messages

If you want to automatically drop unmatched messages (that is, discard every message that does not match a pattern in the pattern database), use the `drop-unmatched()` option in the definition of the pattern database:

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

Note that the `drop-unmatched()` option is available in syslog-ng OSE version 3.11 and later.

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 OSE 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 OSE 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 e-mail server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message. Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages with syslog-ng OSE, 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. Starting with syslog-ng OSE version 3.5, if a message is added to a context, syslog-ng OSE 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 OSE is reloaded (SIGHUP), but are lost when syslog-ng OSE 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 OSE waits for related messages to arrive.

Note the following points about timeout values:

- When a new message is added to a context, syslog-ng OSE 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 OSE 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 OSE 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 OSE will replace its timestamp with the current system time.

  Example: How syslog-ng OSE 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 OSE receives the messages within 1 sec, the timeout event will occour immediately, because the difference of the two timestamp (60 sec) is larger than the timeout value (10 sec).

- Avoid using unnecessarily long timeout values on high-traffic systems, as storing the contexts for many messages can require considerable memory. For example, if two related messages usually arrive within seconds, it is not needed to set the timeout to several hours.

Example: 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 OSE 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 OSE version 3.8 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 e-mail, 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 OSE 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 OSE 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 OSE to send an e-mail alert if a client stops sending messages.

- Configure your clients to send MARK messages periodically. It is enough to configure MARK messages for the destination that forwards your log messages to your syslog-ng OSE server (`mark-mode(periodical)`).
- On your syslog-ng OSE server, create a pattern database rule that matches on the incoming MARK messages. In the rule, set the `context-scope` attribute to `host`, and the `context-timeout` attribute to a value that is higher than the `mark-freq` value set on your clients (by default, `mark-freq` is 1200 seconds, so set `context-timeout` at least to 1500 seconds, but you might want to use a higher value, depending on your environment).
- Add an action to this rule that sends you an e-mail alert if the `context-timeout` expires, and the server does not receive a new MARK message (`<action trigger="timeout">`).
- On your syslog-ng OSE 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 OSE

The following parsers are available in syslog-ng OSE.

@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 e-mail address. The parameter is a set of characters to strip from the beginning and the end of the e-mail address. That way e-mail 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 e-mail 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 (`\xe2\x80\x93`) 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 OSE 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 e-mail 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, &abbrev; 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 OSE version 3.5, if a message is added to a context, syslog-ng OSE 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 OSE 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 OSE 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, &abbrev; 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 OSE 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 OSE 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 OSE 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 OSE 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

- *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 OSE 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 OSE version 3.8 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 OSE version 3.5, if a message is added to a context, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 Open Source 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>
```

16

Correlating log messages

The syslog-ng OSE 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.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix e-mail server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message. Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages with syslog-ng OSE, 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.

Correlating messages using the `grouping-by()` parser

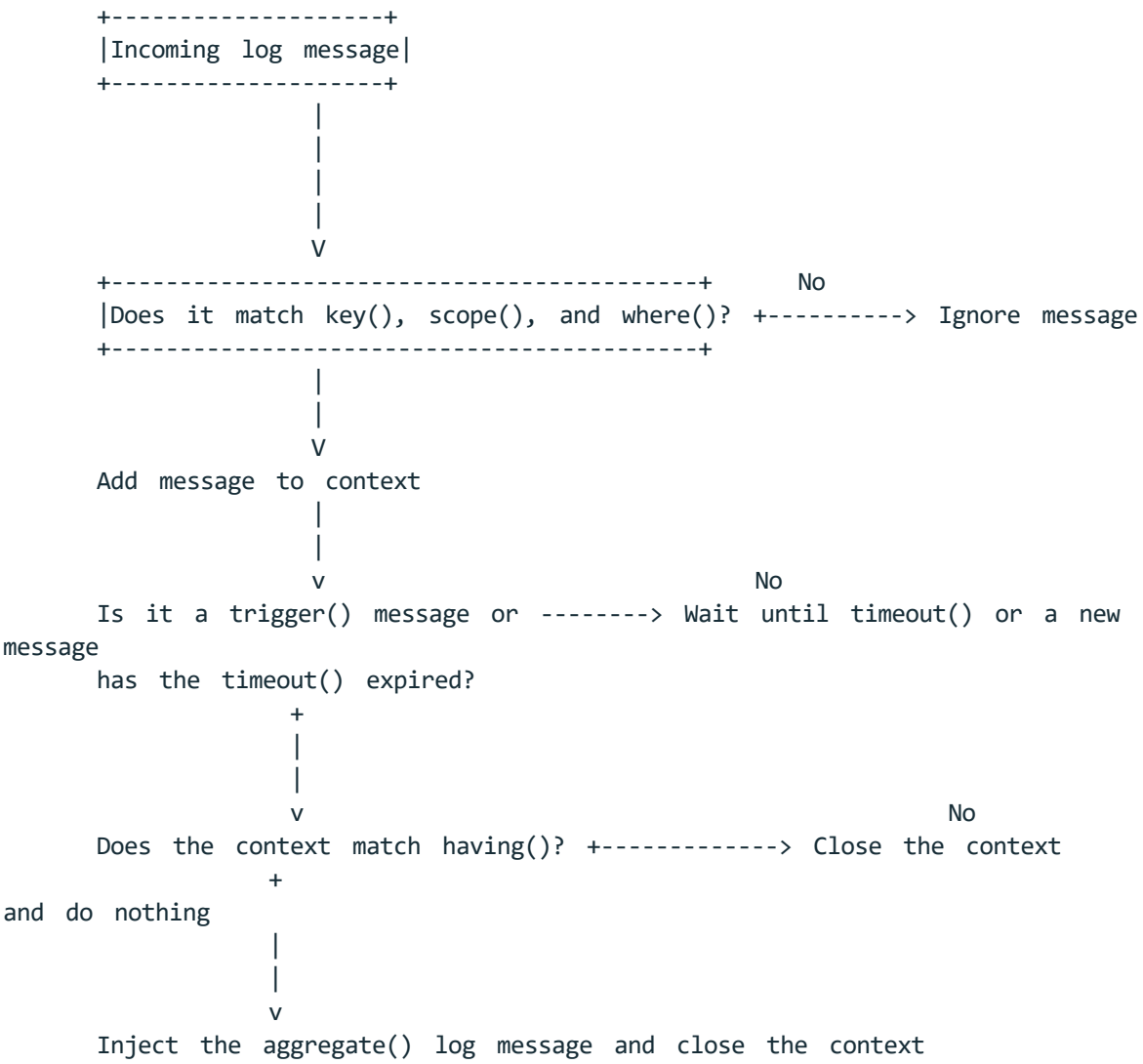
The syslog-ng OSE application can correlate log messages that match a set of filters. This works similarly to SQL GROUP BY statements. Alternatively, you can also correlate log messages using pattern databases. For details, see Correlating log messages using pattern databases.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix e-mail server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the

authentication failure, and the reason of the failure in the next message. Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages with syslog-ng OSE, 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 OSE 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 OSE generates and sends the message set in the `aggregate()` option.

NOTE:

Message contexts are persistent and are not lost when syslog-ng OSE is reloaded (SIGHUP), but are lost when syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 OSE will replace its timestamp with the current system time.

Example: How syslog-ng OSE 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 OSE receives the messages within 1 sec, the timeout event will occour immediately, because the difference of the two timestamp (60 sec) is larger than the timeout value (10 sec).

- Avoid using unnecessarily long timeout values on high-traffic systems, as storing the contexts for many messages can require considerable memory. For example, if two related messages usually arrive within seconds, it is not needed to set the timeout to several hours.

Example: 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=2F62696E2F7368002F7573722F7362696E2F6E7470646174652D64656269616E002D73
```

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 OSE, 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)
       );
};
```

For another example, see The grouping-by() parser in syslog-ng blog post

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}')
 )
```

For another example, see The grouping-by() parser in syslog-ng blog post

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.

Options of `grouping-by` parsers

The `grouping-by` has the following options.

aggregate()

Synopsis:	aggregate()

Description: Specifies the message that syslog-ng OSE 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 OSE 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 OSE 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 OSE 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 as a template (that is, the name of a name-value pair) that every message must have 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.

NOTE:

If the value of the key is static (for example, key("PROGRAM") instead of key ("$PROGRAM")), all messages will 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 OSE 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 OSE 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.

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 OSE 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 OSE 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 GeoIP data from IP addresses (DEPRECATED).
- You can write custom Python modules to process the messages and add data from external files or databases. For details, see The Python Parser.

Adding metadata from an external file

In syslog-ng OSE version 3.8 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 OSE 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 OSE 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 OSE for the changes to take effect. If reloading syslog-ng OSE or the database file fails for some reason, syslog-ng OSE 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 OSE evaluates the filters in the order they appear in the database file. If a filter matches the message, syslog-ng OSE 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 OSE 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 OSE 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 OSE 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: 3.17
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 OSE for the changes to take effect. If reloading syslog-ng OSE or the files fails for some reason, syslog-ng OSE 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 OSE 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 OSE configuration file:

```
add-contextual-data(
       selector("$HOST")
       database("context-info-db.csv")
       default-selector("unknown-hostname")
);
```

ignore-case()

Synopsis:	ignore-case()
Default:	ignore-case(no)

Description: Specifies if selectors are handled as case insensitive. If you set the `ignore-case()` option to **yes**, selectors are handled as case insensitive.

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 OSE evaluates for each message, and if its value matches the ID of an entry in the database, syslog-ng OSE 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 GeoIP data from IP addresses (DEPRECATED)

This parser is deprecated. Use Looking up GeoIP2 data from IP addresses instead.

The syslog-ng OSE application can lookup IPv4 addresses from an offline GeoIP database, and make the retrieved data available in name-value pairs. IPv6 addresses are not supported. Depending on the database used, you can access country code, longitude, and latitude information.

NOTE:

To access longitude and latitude information, download the GeoLiteCity database, and unzip it (for example, to the `/usr/share/GeoIP/GeoLiteCity.dat` file). The default databases available on Linux and other platforms usually contain only the country codes.

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}`.

Declaration:

```
parser  parser_name  {
       geoip(
              <macro-containing-the-IP-address-to-lookup>
              prefix()
              database("<path-to-database-file>")
       );
};
```

Example: Using the GeoIP parser

In the following example, syslog-ng OSE retrieves the GeoIP data of the IP address contained in the ${HOST} field of the incoming message, and includes the data (prefixed with the `geoip.` string) in the output JSON message.

```
@version: 3.7
@module geoip

options {
    keep-hostname(yes);
};

source s_file {
    file("/tmp/input");
};
```

```
parser p_geoip { geoip( "${HOST}", prefix( "geoip." ) database(
"/usr/share/GeoIP/GeoLiteCity.dat" ) ); };

destination d_file {
    file( "/tmp/output" template("$(format-json --scope core --key geoip*)\n")
);
};


log {
    source(s_file);
    parser(p_geoip);
    destination(d_file);
};
```

For example, for the `<38>Jan 1 14:45:22 192.168.1.1 prg00000[1234]: test message` message the output will look like:

```
{"geoip":{"longitude":"47.460704","latitude":"19.049968","country_
code":"HU"},"PROGRAM":"prg00000","PRIORITY":"info","PID":"1234","MESSAGE":"test
message","HOST":"192.168.1.1","FACILITY":"auth","DATE":"Jan  1 14:45:22"}
```

If you are transferring your log messages into Elasticsearch, use the following rewrite rule to combine the longitude and latitude information into a single value (called `geoip.location`), and set the mapping in Elasticsearch accordingly. Do not forget to include the rewrite in your log path. For details on transferring your log messages to Elasticsearch, see elasticsearch: Sending messages directly to Elasticsearch version 1.x.

```
rewrite  r_geoip  {
       set(
             "${geoip.latitude},${geoip.longitude}",
             value(  "geoip.location"  ),
             condition(not  "${geoip.latitude}"  ==  "")
       );
 };
```

In your Elasticsearch configuration, set the appropriate mappings:

```
{
   "mappings" : {
      "_default_" : {
         "properties" : {
            "geoip" : {
               "properties" : {
                  "country_code" : {
                     "index" : "not_analyzed",
                     "type" : "string",
```

```
                  "doc_values" : true
               },
               "latitude" : {
                  "index" : "not_analyzed",
                  "type" : "string",
                  "doc_values" : true
               },
               "longitude" : {
                  "type" : "string",
                  "doc_values" : true,
                  "index" : "not_analyzed"
               },
               "location" : {
                  "type" : "geo_point"
               }
            }
         }
      }
   }
}
```

Options of `geoip` parsers

The `geoip` 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 OSE. 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 vs.

soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

For example, to insert the `geoip.` prefix, use the `prefix(.geoip.)` option. To refer to a particular data when using a prefix, use the prefix in the name of the macro, for example, `${geoip.country_code}` .

database()

Synopsis:	database()
Default:	`/usr/share/GeoIP/GeoIP.dat`

Description: The full path to the GeoIP database to use. Note that syslog-ng OSE must have the required privileges to read this file. Do not modify or delete this file while syslog-ng OSE is running, it can crash syslog-ng OSE.

Looking up GeoIP2 data from IP addresses

The syslog-ng OSE 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 OSE application works with the Country and the City version of the GeoIP2 database, both free and the commercial editions. The syslog-ng OSE 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.

Referring to parts of the message as a macro

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}`.

For example if the default prefix (`.geoip2`) is used, you can determine the country code using `${.geoip2.country.iso_code}`.

To look up all keys:

1. Install the `mmdb-bin` package.

   After installing this package, you will be able to use the `mmdblookup` command.

   > NOTE:
   >
   > The name of the package depends on the Linux distribution. The package mentioned in this example is on Ubuntu.

2. Create a dump using the following command: `mmdblookup --file GeoLite2-City.mmdb --ip <your-IP-address>`

   The resulting dump file will contain the keys that you can use.

For a more complete list of keys, you can also check the GeoIP2 City and Country CSV Databases. However, note that the syslog-ng OSE application works with the `mmdb` (GeoIP2) format of these databases. Other formats, like `csv` are not supported.

Using the GeoIP2 parser

Declaration:

```
parser  parser_name  {
       geoip2(
              <macro-containing-the-IP-address-to-lookup>
              prefix()
              database("<path-to-geoip2-database-file>")
       );
 };
```

In the following example, syslog-ng OSE retrieves the GeoIP2 data of the IP address contained in the ${HOST} field of the incoming message (assuming that in this case the ${HOST} field contains an IP address), and includes the data (prefixed with the `geoip2` string) in the output JSON message.

```
@version: 3.11
@module geoip2

options {
    keep-hostname(yes);
```

```
};

source s_file {
    file("/tmp/input");
};

parser p_geoip2 {
    geoip2(
        "${HOST}",
        prefix( "geoip2." )
        database( "/usr/share/GeoIP2/GeoLiteCity.dat" )
    );
};

destination d_file {
    file(
        "/tmp/output"
        flags(syslog-protocol)
        template("$(format-json --scope core --key geoip2*)\n")
    );
};


log {
    source(s_file);
    parser(p_geoip2);
    destination(d_file);
};
```

For example, for the `<38>2017-05-24T13:09:46 192.168.1.1 prg00000[1234]: test message` message the output will look like:

```
<38>1 2017-05-24T13:09:46+02:00 192.168.1.1 prg00000 1234 - [meta sequenceId="3"]
{"geoip2":{"subdivisions":{"0":{"names":{"en":"Budapest"},"iso_code":"BU","geoname_
id":"3054638"}},"registered_country":{"names":{"en":"Hungary"},"iso_
code":"HU","geoname_id":"719819"},"postal":{"code":"1063"},"location":{"time_
zone":"Europe/Budapest","longitude":"19.070200","latitude":"47.510200","accuracy_
radius":"5"},"country":{"names":{"en":"Hungary"},"iso_code":"HU","geoname_
id":"719819"},"continent":{"names":{"en":"Europe"},"geoname_
id":"6255148","code":"EU"},"city":{"names":{"en":"Budapest"},"geoname_
id":"3054643"}},"PROGRAM":"prg00000","PRIORITY":"info","PID":"1234","MESSAGE":"test
message","HOST":"192.168.1.1","FACILITY":"auth","DATE":"May 24 13:09:46"}
```

Transferring your logs to Elasticsearch using GeoIP2

If you are transferring your log messages into Elasticsearch, use the following rewrite rule to combine the longitude and latitude information into a single value (called

geoip2.location), and set the mapping in Elasticsearch accordingly. Do not forget to include the rewrite in your log path. These examples assume that you used prefix("geoip2.") instead of the default for the geoip2 parser. For details on transferring your log messages to Elasticsearch, see elasticsearch2: Sending logs directly to Elasticsearch and Kibana 2.0 or higher.

```
rewrite r_geoip2 {
       set(
             "${geoip2.location.latitude},${geoip2.location.longitude}",
             value( "geoip2.location2" ),
             condition(not "${geoip2.location.latitude}" == "")
       );
};
```

In your Elasticsearch configuration, set the appropriate mappings:

```
{
     "mappings" : {
          "_default_" : {
               "properties" : {
                    "geoip2" : {
                         "properties" : {
                              "location2" : {
                                   "type" : "geo_point"
                              }
                         }
                    }
               }
          }
     }
}
```

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 OSE. 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 vs. soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

For example, to insert the `.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 OSE must have the required privileges to read this file. Do not modify or delete this file while syslog-ng OSE is running, it can crash syslog-ng OSE.

18

Statistics of syslog-ng

The syslog-ng OSE 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 OSE. The `stats-level()` global option determines exactly which statistics syslog-ng OSE collects. You can access these statistics and metrics using the following methods.

Recommended: Structured, selective methods:

- Using the `syslog-ng-ctl query` command.

Legacy: Unstructured, bulk methods:

- Using the `internal()` source.
- Using the `syslog-ng-ctl stats` command.
- 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 OSE

You can list all active metrics on your syslog-ng OSE 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)`
     - `sender`: sender host, from `stats-level(3)`
     - `program`: program, 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 OSE, 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 OSE application stores the statistics of the objects when syslog-ng OSE is reloaded. However, if the configuration of syslog-ng OSE was changed since the last reload, the statistics of orphaned objects are deleted.
5. The type of the statistics:

- `processed`: The number of messages that successfully reached their destination driver. Note that this does not necessarily mean that the destination driver successfully delivered the messages (for example, written to disk or sent to a remote server).
- `dropped`: The number of dropped messages — syslog-ng OSE 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 OSE dropped the message for some other reason (for example, a parsing error).
- `queued`: The number of messages passed to the message queue of the destination driver, waiting to be sent to the destination.
- `suppressed`: The number of suppressed messages (if the `suppress()` feature is enabled).
- `stamp`: The UNIX timestamp of the last message sent to the destination.
- `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
  ```

- `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 that the memory usage (size) of queues does not equal to the memory usage (size) of the log messages in syslog-ng OSE. 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.
- `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
  ```

- `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 that 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
  ```

- `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 OSE 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 OSE 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.

6. 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.

Log statistics from the `internal()` source

If the `stats-freq()` global option is higher than 0, syslog-ng OSE 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`

Multithreading and scaling in syslog-ng OSE

Starting with version 3.3, syslog-ng OSE can process sources and destinations in multithreaded mode to scale to multiple CPUs or cores for increased performance. Starting with version 3.6, this multithreaded mode is the default.

Multithreading concepts of syslog-ng OSE

This section is a brief overview on how syslog-ng OSE 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 OSE uses multithreading may change in future releases. The current documentation applies to version 3.17.

syslog-ng OSE 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 OSE.

The maximum number of worker threads syslog-ng OSE uses is the number of CPUs or cores in the host running syslog-ng OSE (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 OSE can use, including the main syslog-ng OSE thread. However, the `--worker-threads` option does not affect the supervisor of syslog-ng OSE. 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 OSE.

When an event requiring a new thread occurs (for example, syslog-ng OSE receives new messages, or a destination becomes available), syslog-ng OSE 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 OSE will use separate threads for these sources even if they both have only a single active connection.
- The `udp`, `file`, and `pipe` sources use a single thread for every source statement.
- The `tcp`, `syslog`, and `pipe` destinations use a single thread for every destination.
- The `file` destination uses a single thread for writing the destination file, but may use a separate thread for each destination file if the filename includes macros.

Sources and destinations not affected by multithreading

The following list describes sources and destinations that use a separate thread even if you disable multithreading in syslog-ng OSE, in addition to the limit set in 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 3.6, syslog-ng OSE runs in multithreaded mode by default. You can enable multithreading in syslog-ng OSE 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

Destinations that have a queue process that queue in a single thread. Multiple sources can send messages to the same queue, so the queue can scale to multiple CPUs. However, when the writer thread writes the queue contents to the destination, it will be single-threaded.

Message parsing, rewrite rules, filters, and other types of message processing is performed by the reader thread in a sequential manner. This means that such operations can scale only if reading messages from the source can be multithreaded. For example, if a `tcp` source can process messages from different connections (clients) in separate threads. If the source cannot use multiple threads to process the messages, the operations will not scale.

To improve the processing power of syslog-ng OSE and scale to more processors, use the following methods:

- To improve scaling on the source side, use more sources, for example, more source files, or receive the messages from more parallel connections. For network sources, you can also configure a part of your clients to send the messages to a different port of your syslog-ng server, and use separate source definitions for each port.
- On the destination side, 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.
- On the destination side, 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 separate destinations on the clients to use both ports.

20

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 OSE server does not receive the messages, verify that the IP addresses and ports are correct in your sources and destinations. Also, check that the client and the server uses the same protocol (a common error is to send logs on UDP, but configure the server to receive logs on TCP.

  If the problem persist, use `tcpdump` or a similar packet sniffer tool on the client to verify that the messages are sent correctly, and on the server to verify that it receives the messages.
- To find message-routing problems, run syslog-ng OSE with the following command `syslog-ng -Fevd`. That way syslog-ng OSE will run in the foreground, and display debug messages about the messages that are processed.
- If syslog-ng is closing the connections for no apparent reason, be sure to check the log messages of syslog-ng. You might also want to run syslog-ng with the `--verbose` or `--debug` command-line options for more-detailed log messages. You can enable these messages without restarting syslog-ng using the `syslog-ng-ctl verbose --set=on` command. For details, see the syslog-ng-ctl man page at 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 OSE client and the source of your syslog-ng OSE server, everything should work as expected. Unfortunately there are some other combinations, that seem to be working, but result in losing parts of the messages. For details on the working combinations, see Things to consider when forwarding messages between syslog-ng OSE hosts.
- In case you experience a problem that is not covered in this guide, send it to our mailing list.

  To report bugs found in syslog-ng OSE, visit our GitHub issues page.

Precompiled binary packages are available for free from various third-parties. See the list of precompiled syslog-ng OSE binary packages.

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.

- *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 Statistics 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.
- *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 Statistics of syslog-ng).
- *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 OSE is writing messages into files*: If syslog-ng OSE 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

Purpose:

When syslog-ng crashes for some reason, it can create a core file that contains important troubleshooting information. To enable core files, complete the following procedure:

Steps:

1. Core files are produced only if the `maximum core file size` ulimit is set to a high value in the init script of syslog-ng.Add the following line to the init script of syslog-ng:

   ```
   ulimit -c unlimited
   ```

2. Verify that syslog-ng has permissions to write the directory it is started from, for example `/opt/syslog-ng/sbin/`.
3. If syslog-ng crashes, it will create a core file in the directory syslog-ng was started from.
4. To test that syslog-ng can create a core file, you can create a crash manually. For this, determine the PID of syslog-ng (for example using the `ps -All|grep syslog-ng` command), then issue the following command: `kill -ABRT <syslog-ng pid>`

   This should create a core file in the current working directory.

Collecting debugging information with strace, truss, or tusc

To properly troubleshoot certain situations, it can be useful to trace which system calls syslog-ng OSE performs. How this is performed depends on the platform running syslog-ng OSE. In general, note the following points:

- When syslog-ng OSE 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 may be different for your installation, as 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 OSE process, use the `-p <pid_of_syslog-ng>` parameter.

Running a failure script

Purpose:

You can create a failure script that is executed when syslog-ng OSE terminates abnormally, that is, when it exits with a non-zero exit code. For example, you can use this script to send an automatic e-mail notification.

Prerequisites:

The failure script must be the following file: `/opt/syslog-ng/sbin/syslog-ng-failure`, and must be executable.

To create a sample failure script, complete the following steps.

Steps:

1. Create a file named `/opt/syslog-ng/sbin/syslog-ng-failure` with the following content:

```
#!/usr/bin/env bash
cat >>/tmp/test.txt <<EOF
$(date)
Name.....$1
```

```
Chroot dir......$2
Pid file dir....$3
Pid file........$4
Cwd.............$5
Caps............$6
Reason..........$7
Argbuf..........$8
Restarting......$9

EOF
```

2. Make the file executable: `chmod +x /opt/syslog-ng/sbin/syslog-ng-failure`
3. Run the following command in the `/opt/syslog-ng/sbin` directory: `./syslog-ng --process-mode=safe-background; sleep 0.5; ps aux | grep './syslog-ng' | grep -v grep | awk '{print $2}' | xargs kill -KILL; sleep 0.5; cat /tmp/test.txt`

   The command starts syslog-ng OSE 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 OSE 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-ng-debun` tool to collect information about your environment and syslog-ng OSE version. For details, see The syslog-ng-debun manual page. For support contacts, see About us.

Recover data from orphaned diskbuffer files

When you change the configuration of a syslog-ng OSE host that uses disk-based buffering (also called disk queue), syslog-ng OSE may start new disk buffer files for the destinations that you have changed. In such case, syslog-ng OSE abandons the old disk queue files. If there were unsent log messages in the disk queue files, these messages remain in the disk queue files, and will not be sent to the destinations.

No local logs after specifying an unusual storage directory

Security-Enhanced Linux (SELinux) is a set of kernel and user-space tools enforcing strict access control policies. SELinux rules in Linux distributions cover all aspects of the syslog-ng configuration coming in the syslog-ng package available in the distribution. But as soon as an unusual port number or directory name is specified in the configuration, syslog-ng fails to work even with a completely legitimate configuration.

When you choose to save logs of a central syslog-ng OSE server to a directory other than the `/var/log` directory, logs will not start appearing on the newly configured directory. For details on how to fix this issue, see sectionUsing a different storage directoryin the blog post titled Using syslog-ng with SELinux in enforcing mode.

No logs after specifying an unusual port number

Security-Enhanced Linux (SELinux) is a set of kernel and user-space tools enforcing strict access control policies. SELinux rules in Linux distributions cover all aspects of the syslog-ng configuration coming in the syslog-ng package available in the distribution. But as soon

as an unusual port number or directory name is specified in the configuration, syslog-ng fails to work even with a completely legitimate configuration.

By default, SELinux only allows connections to the default syslog ports. When you have to use any other port for some reason, sending logs to that port will not work. For details on how to fix this issue, see sectionUsing a different portin the blog post titled Using syslog-ng with SELinux in enforcing mode.

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

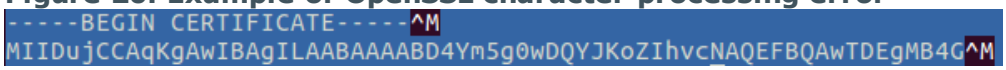
Error message:
```
testuser@thor-x1:~/cert_no_start_line/certs$ openssl x509 -in cert.pem -
text
unable to load certificate
140178126276248:error:0906D06C:PEM routines:PEM_read_bio:no start
line:pem_lib.c:701:Expecting: TRUSTED CERTIFICATE
```

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^Mcharacters as shown in the image below:

Figure 20: 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.

21

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 subsequent sections.

- Do not base the separation of log messages into different files on the `facility` parameter. As several applications and processes can use the same facility, the facility does not identify the application that sent the message. By default, the `facility` parameter is not even included in the log message itself. In general, sorting the log messages into several different files can make finding specific log messages difficult. If you must create separate log files, use the application name.
- Standard log messages include the local time of the sending host, without any time zone information. It is recommended to replace this timestamp with an ISODATE timestamp, because the ISODATE format includes the year and timezone as well. To convert all timestamps to the ISODATE format, include the following line in the syslog-ng configuration file:

  ```
  options {ts-format(iso) ; };
  ```

- Resolving the IP addresses of the clients to domain names can decrease the performance of syslog-ng. For details, see 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 OSE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

  As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.
- Increase the value of the `flush-lines()` parameter. Increasing `flush-lines()` from `0` to `100` can increase the performance of syslog-ng OSE by 100%.

Using name resolution in syslog-ng

The syslog-ng application can resolve the hostnames of the clients and include them in the log messages. However, the performance of syslog-ng is severely degraded if the domain name server is unaccessible or slow. Therefore, it is not recommended to resolve hostnames in syslog-ng. If you must use name resolution from syslog-ng, consider the following:

- Use DNS caching. Verify that the DNS cache is large enough to store all important hostnames. (By default, the syslog-ng DNS cache stores `1007` entries.)

  ```
  options { dns-cache-size(2000); };
  ```
- If the IP addresses of the clients change only rarely, set the expiry of the DNS cache large.

  ```
  options { dns-cache-expire(87600); };
  ```
- If possible, resolve the hostnames locally. For details, see Resolving hostnames locally.

NOTE:

Domain name resolution is important mainly in relay and server mode.

Resolving hostnames locally

Purpose:

Resolving hostnames locally enables you to display hostnames in the log files for frequently used hosts, without having to rely on a DNS server. The known IP address \xe2\x80\x93 hostname pairs are stored locally in a file. In the log messages, syslog-ng will replace the IP addresses of known hosts with their hostnames. To configure local name resolution, complete the following steps:

Steps:

1. Add the hostnames and the respective IP addresses to the file used for local name resolution. On Linux and UNIX systems, this is the `/etc/hosts` file. Consult the documentation of your operating system for details.
2. Instruct syslog-ng to resolve hostnames locally. Set the `use-dns()` option of syslog-ng to `persist_only`.
3. Set the `dns-cache-hosts()` option to point to the file storing the hostnames.

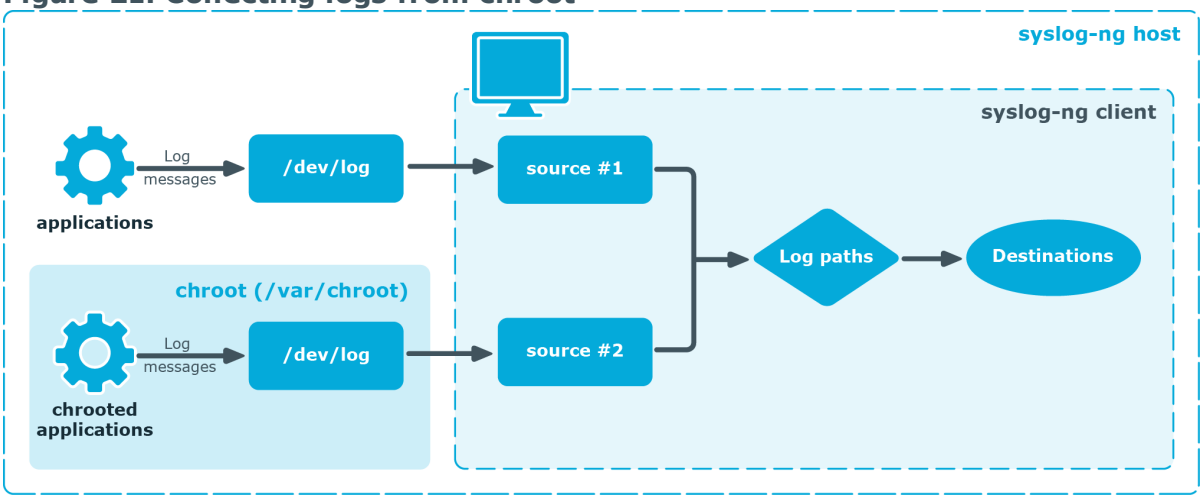
```
options  {
       use-dns(persist_only);
       dns-cache-hosts(/etc/hosts);
 };
```

Collecting logs from chroot

Purpose:

To collect logs from a chroot using a syslog-ng client running on the host, complete the following steps:

Figure 21: Collecting logs from chroot



Steps:

1. Create a `/dev` directory within the chroot. The applications running in the chroot send their log messages here.
2. Create a local source in the configuration file of the syslog-ng application running outside the chroot. This source should point to the `/dev/log` file within the chroot (for example to the `/chroot/dev/log` directory).
3. Include the source in a log statement.

   NOTE:

   You need to set up timezone information within your chroot as well. This usually means creating a symlink to `/etc/localtime`.

Configuring log rotation

The syslog-ng OSE application does not rotate logs by itself. To use syslog-ng OSE for log rotation, consider the following approaches:

Use logrotate together with syslog-ng OSE:

- 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 OSE must be reloaded/restarted when the files are rotated). After rotating the log files, reload syslog-ng OSE using the `syslog-ng-ctl reload` command, or use another method to send a SIGHUP to syslog-ng OSE.

- The statistics collected by syslog-ng OSE, and the correlation information gathered with Pattern Database is lost with each restart.

Separate incoming logs based on time, host or other information:

- Ideal for central log servers, where regular restart of syslog-ng OSE 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). (For details on using macros, see Templates and macros.)

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)
       );
};
```

This sample logstore destination configuration stores incoming logs in logstores that are named based on the current year, month and day, and places these logstores in directories that are named based on the hostname:

```
destination d_logstore {
       logstore(
              "/var/log/remote/${HOST}/${YEAR}_${MONTH}_${DAY}.lgs"
              compress(9)
              create-dirs(yes)
       );
};
```

Example: 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 {} \;
```

22

The syslog-ng manual pages

This chapter collects the manual pages of syslog-ng OSE and other related applications that are usually distributed and packaged together with the syslog-ng Open Source Edition application.

The dqtool tool manual page

Table of Contents

dqtool — Display the contents of a disk-buffer file created with syslog-ng Open Source Edition

Name

dqtool — Display the contents of a disk-buffer file created with syslog-ng Open Source Edition

Synopsis

`dqtool` [command] [options]

Description

NOTE: The dqtool application is distributed with the syslog-ng Open Source Edition system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at the 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.lgs |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
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPA
DDPADDPADDPADDPADD
Mar  3 10:52:05 tristram localprg[1234]: seq: 0000011631, runid: 1267609923, stamp:
2010-03-03T10:52:05
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPA
DDPADDPADDPADDPADD
```

Files

```
/opt/syslog-ng/bin/dqtool
```

See also

syslog-ng.conf(5)

syslog-ng(8)

Note

For the detailed documentation of syslog-ng OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

The loggen manual page

Table of Contents

loggen — Generate syslog messages at a specified rate

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.

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

--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 OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

The pdbtool manual page

Table of Contents

pdbtool — An application to test and convert syslog-ng pattern database rules

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 MESSAGE 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 ssh2* message:

```
pdbtool match -p patterndb.xml -P sshd -M "Accepted publickey for myuser from
127.0.0.1 port 59357 ssh2"
```

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 syslog-ng Documentation page 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 syslog-ng Documentation page. The **patternize** command is available only in syslog-ng OSE 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 OSE 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 Documentation page

syslog-ng.conf(5)

syslog-ng(8)

Note

For the detailed documentation of syslog-ng OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

The syslog-ng control tool manual page

Table of Contents

syslog-ng-ctl — Display message statistics and enable verbose, debug and trace modes in syslog-ng Open Source Edition

Name

syslog-ng-ctl — Display message statistics and enable verbose, debug and trace modes in syslog-ng Open Source Edition

Synopsis

`syslog-ng-ctl` [command] [options]

Description

NOTE: The syslog-ng-ctl application is distributed with the syslog-ng Open Source Edition system logging application, and is usually part of the syslog-ng package. The latest version of the syslog-ng application is available at 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
- reload the configuration of syslog-ng OSE.

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 OSE 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 OSE 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 OSE 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 OSE 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
```

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 OSE 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 OSE 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 OSE 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 OSE configuration will be unaffected. You must provide the passphrase of the password-protected keys every time syslog-ng OSE 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 OSE 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 OSE configuration file.

--secret=<passphrase-of-the-key> or **-s**

The password or passphrase of the private key.

Reloading the configuration

`syslog-ng-ctl reload` [options]

Use the **syslog-ng-ctl reload** command to reload the configuration file of syslog-ng OSE without having to restart the syslog-ng OSE application. The **syslog-ng-ctl reload** works like a SIGHUP.

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 OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

The syslog-ng-debun manual page

Table of Contents

syslog-ng-debun — syslog-ng DEBUg buNdle generator

Name

syslog-ng-debun — syslog-ng DEBUg buNdle generator

Synopsis

`syslog-ng-debun` [options]

Description

NOTE: The **syslog-ng-debun** application is distributed with the syslog-ng OSE system logging application, and is usually part of the syslog-ng OSE package. The latest version of the syslog-ng OSE 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-ng-debun** tool collects and saves information about your syslog-ng OSE installation, making troubleshooting easier, especially if you ask help about your syslog-ng OSE related problem.

General Options

-r

Run **syslog-ng-debun**. Using this option is required to actually execute the data collection with **syslog-ng-debun**. It is needed to prevent accidentally running **syslog-ng-debun**.

-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 OSE 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 OSE 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 OSE 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 OSE 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 OSE 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-ng-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 OSE in noninteractive debug mode for <seconds>, and automatically exit debug mode after the specified number of seconds.

Examples

```
syslog-ng-debun -r
```

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-ng-debun -r -l
```

Similar to **syslog-ng-debun -r**, 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-ng-debun -r -d
```

Similar to **syslog-ng-debun -r**, 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-ng-debun -r -s
```

Trace the system calls (using **strace** or **truss**) of an already running syslog-ng OSE process.

```
syslog-ng-debun -r -d -s
```

Restart syslog-ng OSE in debug mode, and also trace the system calls (using **strace** or **truss**) of the syslog-ng OSE process.

```
syslog-ng-debun -r -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-ng-debun -r -p -t 10
```

Noninteractive debug mode: Similar to **syslog-ng-debun -r -p**, but automatically exit after 10 seconds.

```
syslog-ng-debun -r -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-ng-debun -r -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 OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

The syslog-ng manual page

Table of Contents

syslog-ng — syslog-ng system logger application

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 Documentation page or the syslog-ng page.

The syslog-ng OSE 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 OSE process with the specified POSIX capability flags.

- If the *--no-caps* option is not set,syslog-ng OSE has been compiled with the *--enable-linux-caps* compile option, and the host supports CAP_SYSLOG, syslog-ng OSE 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 OSE 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 OSE configuration file. The following modules are loaded by default: affile, afprog, afsocket, afuser, basicfuncs, csvparser, dbparser, syslogformat, afsql, system-source. Available only in syslog-ng Open Source Edition 3.3 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 OSE starts from the current directory (**$CWD**) so it can create core files (normally, syslog-ng OSE starts from `$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 Open Source Edition 3.3 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 OSE 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 Open Source Edition 3.3 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 OSE can use, including the main syslog-ng OSE thread. Note that certain operations in syslog-ng OSE can use threads that are not limited by this option. This setting has effect only when syslog-ng OSE is running in multithreaded mode. Available only in syslog-ng Open Source Edition 3.3 and later. See **The syslog-ng Open Source Edition 3.15 Administrator Guide** for details.

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 OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

The syslog-ng.conf manual page

Table of Contents

syslog-ng.conf — syslog-ng configuration file

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 Documentation page or the syslog-ng page.

The syslog-ng OSE 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 OSE

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 OSE 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.
wildcard-file()	Reads messages from multiple files and directories.
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.
nodejs()	Receives JSON messages from nodejs applications.
mbox()	Read e-mail messages from local mbox files, and convert them to multiline log messages.
osquery()	Run osquery queries, and convert their results into log messages.
pacct()	Reads messages from the process accounting logs on Linux.
pipe()	Opens the specified named pipe and reads messages.
program()	Opens the specified application and reads messages from its standard output.
snmptrap()	Read and parse the SNMP traps of the Net-SNMP's snmptrapd application.
sun-stream(), sun-streams()	Opens the specified *STREAMS* device on Solaris systems and reads incoming messages.
syslog()	Listens for incoming messages using the new IETF-standard syslog protocol.

Name	Description
system()	Automatically detects which platform syslog-ng OSE 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.
stdin()	Collects messages from the standard input stream.

Table 2. Destination drivers available in syslog-ng

Name	Description
amqp()	Publishes messages using the AMQP (Advanced Message Queuing Protocol).
elasticsearch and elasticsearch2	Sends messages to an Elasticsearch server. The *elasticsearch2* driver supports Elasticsearch version 2 and newer.
file()	Writes messages to the specified file.
graphite()	Sends metrics to a Graphite server to store numeric time-series data.
graylog2()	Sends syslog messages to Graylog.
hdfs()	Sends messages into a file on a Hadoop Distributed File System (HDFS) node.
http()	Sends messages over the HTTP protocol. There are two different implementations of this driver: a Java-based http driver, and an http driver without Java.
kafka()	Publishes log messages to the Apache Kafka message bus, where subscribers can access them.
loggly()	Sends log messages to the Loggly Logging-as-a-Service provider.
logmatic()	Sends log messages to the Logmatic.io Logging-as-a-Service provider.
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.
redis()	Sends messages as name-value pairs to a Redis key-value store.
riemann()	Sends metrics or events to a Riemann monitoring system.
smtp()	Sends e-mail messages to the specified recipients.
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 ???.
stomp()	Sends messages to a STOMP server.

Name	Description
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 OSE

Name	Description
facility()	Filter messages based on the sending facility.
filter()	Call another filter function.
host()	Filter messages based on the sending host.
inlist()	File-based whitelisting and blacklisting.
level() or priority()	Filter messages based on their priority.
match()	Use a regular expression to filter messages based on a specified header or content field.
message()	Use a regular expression to filter messages based on their content.
netmask()	Filter messages based on the IP address of the sending host.
program()	Filter messages based on the sending application.
source()	Select messages of the specified syslog-ng OSE 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 OSE see the syslog-ng Documentation page

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 Balabit Documentation Team <documentation@balabit.com>.

Copyright

The authors grant permission to copy, distribute and/or modify this manual page under the terms of the GNU General Public License Version 2 or newer (GPL v2+).

23

Third-party contributions

This appendix includes the open source licenses and attributions applicable to syslog-ng Open Source Edition.

GNU General Public License

Version 2, June 1991

1989, 1991 Free Software Foundation, Inc.

Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Section 1 and Section 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by

modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type "show w". This is free software, and you are welcome to redistribute it under certain conditions; type "show c" for details.

The hypothetical commands "show w" and "show c" should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than "show w" and "show c" ; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program "Gnomovision" (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to

permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

GNU Lesser General Public License

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.

Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method:

1. we copyright the library, and
2. we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the Lesser General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

Section 1

You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms ofSection 1 above, provided that you also meet all of these conditions:

a. The modified work must itself be a software library.

b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

Section 4

You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Section 1 and Section 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Section 1 and Section 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

Section 5

A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

Section 6

As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Section 1 and Section 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

Section 7

You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

Section 8

You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 9

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

Section 10

Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

Section 11

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 12

If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 13

The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

Section 14

If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 15

BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

NO WARRANTY Section 16

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

License attributions

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (https://www.openssl.org/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)

24

Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED. BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. *Definitions*
    a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
    b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. "Distribute" means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected

performance or phonogram in digital form or other electronic medium.

2. *Fair Dealing Rights.* Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.
3. *License Grant.* Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:
    a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,
    b. to Distribute and Publicly Perform the Work including as incorporated in Collections.

    The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).
4. *Restrictions.* The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
    a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.
    b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (for example a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

    i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

    ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

    iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort,

mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

5. *Representations, Warranties and Disclaimer* UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.
6. *Limitation on Liability.* EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
7. *Termination*
    a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
    b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.
8. *Miscellaneous*
    a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
    b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
    c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
    d. This License constitutes the entire agreement between the parties with respect

to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

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 or other inquiries, visit https://www.oneidentity.com/company/contact-us.aspx or call +1-800-306-9329.

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