

Axoflow

AxoSyslog

Version 4.29.0

16 November 2026

© 2023–2026 Axoflow Inc. All rights reserved.

Built from main 15d7c009

Contents

1 What's new 16

2 Introduction to AxoSyslog 30

3 Quick-start guide 36
- 3.1 Configuring AxoSyslog on client hosts 37
- 3.2 Configuring AxoSyslog as a logserver 42
- 3.3 Configuring AxoSyslog relays 47
  - 3.3.1 Configuring AxoSyslog on relay hosts 48
  - 3.3.2 How relaying log messages works 51
- 3.4 Sending Kubernetes logs to OpenSearch 52
- 3.5 Managing and checking the syslog-ng service on Linux 63

4 Install AxoSyslog 67
- 4.1 Install AxoSyslog with Podman and systemd 69
- 4.2 Install AxoSyslog on Debian/Ubuntu 75
- 4.3 Install AxoSyslog on RHEL/Fedora/AlmaLinux 80
- 4.4 Install AxoSyslog with Helm 88
  - 4.4.1 Parameters of the AxoSyslog Helm chart 94
- 4.5 Install AxoSyslog with Podman 113
- 4.6 Install AxoSyslog with Docker 117
- 4.7 AxoSyslog on macOS 121
- 4.8 Upgrade AxoSyslog to a newer version 124
- 4.9 Upgrade syslog-ng to AxoSyslog 135
- 4.10 Forward Windows logs 138

5 The concepts of AxoSyslog 140
- 5.1 The philosophy of AxoSyslog 141
- 5.2 Logging with AxoSyslog 142
  - 5.2.1 The route of a log message in AxoSyslog 143
- 5.3 Modes of operation 145
  - 5.3.1 Client mode 146
  - 5.3.2 Relay mode 147
  - 5.3.3 Server mode 150
- 5.4 Global objects 151
- 5.5 Timezones and daylight saving 153
  - 5.5.1 Assigning timezone to the message 154
  - 5.5.2 A note on timezones and timestamps 156
- 5.6 Product licensing 157
- 5.7 High availability support 158

5.8 The structure of a log message 159
5.8.1 BSD-syslog or legacy-syslog messages 160
5.8.2 IETF-syslog messages 162
5.8.3 Enterprise-wide message model (EWMM) 165
5.8.4 The PRI message part 167
5.9 Message representation in AxoSyslog 171
5.10 Structuring macros, metadata, and other value-pairs 173
5.10.1 Specifying data types in value-pairs 174
5.10.2 value-pairs() 178
5.11 Things to consider when forwarding messages between AxoSyslog hosts 185

6 The configuration file 188
6.1 Location of the configuration file 189
6.2 The configuration syntax in detail 190
6.3 Notes about the configuration syntax 194
6.4 Defining configuration objects inline 195
6.5 Using channels in configuration objects 196
6.6 Global and environmental variables 198
6.7 Configuration identifier 200
6.8 Using modules 201
6.8.1 Loading modules 202
6.8.2 Listing configuration options 204
6.8.3 Visualize the configuration 206
6.9 Managing complex configurations 207
6.9.1 Including configuration files 208
6.9.2 Reusing configuration blocks 210
6.9.3 Generating configuration blocks from a script 216
6.10 Python code in external files 219
6.11 Logging from your Python code 222

7 source: Read, receive, and collect log messages 224
7.1 How sources work 225
7.2 Arr logs 231
7.3 axosyslog-otlp(): Receive logs from another node using OpenTelemetry 233
7.4 Collect native macOS system logs 254
7.5 default-network-drivers: Receive and parse common syslog messages 272
7.5.1 default-network-drivers() source options 275
7.6 ehttp: Receive logs over HTTP 286
7.7 elasticsearch-bulk: Receive messages from Elastic clients 312
7.8 file: Collect messages from text files 332
7.8.1 Notes on reading kernel messages 334
7.8.2 file() source options 335

7.9 Hypr Audit Trail and Hypr App Audit Trail 360
7.10 internal: Collect internal messages 363
7.11 Jellyfin logs 366
7.12 kubernetes: Collect and parse the Kubernetes CRI (Container Runtime Interface) format 368
7.13 linux-audit: Collect messages from Linux audit logs 371
7.14 mbox: Convert local email messages to log messages 374
7.15 mqtt: receiving messages from an MQTT broker 377
7.15.1 Options of the mqtt() source 380
7.16 network: Collect messages using the RFC3164 protocol (network() driver) 397
7.16.1 network() source options 400
7.16.2 Proxy Protocol support 430
7.16.2.1 The working mechanism behind the Proxy Protocol 431
7.16.2.2 Proxy Protocol: configuration and output examples 433
7.17 nodejs: Receive JSON messages from nodejs applications 436
7.18 Collect OpenBSD kernel log messages 440
7.19 Receive logs, metrics, and traces from OpenTelemetry 444
7.20 osquery: Collect and parse osquery result logs 466
7.20.1 osquery() source options 470
7.21 pacct: Collect process accounting logs on Linux 473
7.22 Pi-hole Faster Than Light logs 477
7.23 pipe: Collect messages from named pipes 478
7.23.1 pipe() source options 479
7.24 program: Receive messages from external applications 504
7.24.1 program() source options 505
7.25 python: writing server-style Python sources 524
7.25.1 Python LogMessage API 531
7.25.2 python() and python-fetcher() source options 534
7.26 python-fetcher: writing fetcher-style Python sources 552
7.27 qBittorrent logs 558
7.28 snmptrap: Read Net-SNMP traps 559
7.28.1 snmptrap() source options 564
7.29 splunk-hec: Receive messages from Splunk HEC clients 569
7.30 stdin: Collect messages from the standard input stream 590
7.30.1 stdin() source options 591
7.31 sun-streams: Collect messages on Sun Solaris 614
7.31.1 sun-streams() source options 616
7.32 syslog: Collect messages using the IETF-syslog protocol 629
7.32.1 syslog() source options 631
7.33 system: Collect the system-specific log messages of a platform 661
670

7.34.1 systemd-journal() source options 674
7.35 systemd-syslog: Collect systemd messages using a socket 687
7.35.1 systemd-syslog() source options 688
7.36 tcp, tcp6, udp, udp6: OBSOLETE - Collect messages from remote hosts using the BSD syslog protocol 708
7.36.1 tcp(), tcp6(), udp() and udp6() source options: OBSOLETE 709
7.36.1.1 Change an old source driver to the network() driver 710
7.37 unix-stream, unix-dgram: Collect messages from UNIX domain sockets 712
7.37.1 UNIX credentials and other metadata 714
7.37.2 unix-stream() and unix-dgram() source options 715
7.38 Webhook 739
7.39 wildcard-file: Collect messages from multiple text files 747
7.39.1 wildcard-file() source options 749

8 destination: Forward, send, and store log messages 777

8.1 amqp: Publish messages using AMQP 779
8.1.1 amqp() destination options 781
8.2 Send data to Apache Arrow Flight 800
8.3 Send data to Azure Monitor and Sentinel 818
8.4 Send data to Google BigQuery 823
8.5 ClickHouse database 849
8.6 collectd: Send metrics to collectd 879
8.6.1 collectd() destination options 881
8.7 discord: Send alerts and notifications to Discord 895
8.7.1 Discord destination options 897
8.8 elasticsearch2: DEPRECATED - Send messages directly to Elasticsearch version 2.0 or higher 920
8.8.1 Prerequisites 925
8.8.2 How AxoSyslog interacts with Elasticsearch 926
8.8.3 Client modes 927
8.8.4 Search Guard 929
8.8.5 Elasticsearch2 destination options (DEPRECATED) 931
8.9 Send messages to Elasticsearch data streams 960
8.10 elasticsearch-http: Send messages to Elasticsearch HTTP Bulk API 962
8.10.1 Batch mode and load balancing 965
8.10.2 elasticsearch-http() destination options 968
8.11 file: Store messages in plain-text files 992
8.11.1 file() destination options 994
8.12 Google Pub/Sub gRPC 1013
8.13 Google Pub/Sub HTTP REST API 1038
8.14 graphite: Send metrics to Graphite 1044
8.14.1 graphite() destination options 1046

8.15 graylog2: Send logs to Graylog 1050
8.15.1 graylog2() destination options 1052
8.16 hdfs: Store messages on the Hadoop Distributed File System (HDFS) 1056
8.16.1 Prerequisites 1059
8.16.2 How AxoSyslog interacts with HDFS 1060
8.16.3 Storing messages with MapR-FS 1062
8.16.4 Kerberos authentication with the hdfs() destination 1064
8.16.5 HDFS destination options 1066
8.17 java: Post messages over HTTP using Java 1082
8.17.1 HTTP destination options 1084
8.18 http: Post messages over HTTP without Java 1091
8.18.1 Batch mode and load balancing 1094
8.18.2 HTTP destination options 1098
8.18.3 The Azure auth header plugin 1140
8.18.4 The Python HTTP header plugin 1141
8.19 kafka: Publish messages to Apache Kafka 1146
8.19.1 Prerequisites 1148
8.19.2 How AxoSyslog interacts with Apache Kafka 1149
8.19.3 Kafka destination options 1150
8.20 kafka-c(): Publish messages to Apache Kafka (C implementation) 1159
8.20.1 Shifting from Java implementation to C implementation 1160
8.20.2 Before you begin 1161
8.20.3 Flow control and the Kafka client 1162
8.20.4 Options of the kafka() destination's C implementation 1163
8.21 loggly: Send logs to Loggly 1182
8.21.1 loggly() destination options 1184
8.22 logmatic: Send logs to Logmatic.io 1188
8.22.1 logmatic() destination options 1190
8.23 Send messages to Falcon LogScale 1193
8.24 loki: Grafana Loki 1197
8.25 mongodb(): Store messages in a MongoDB database 1215
8.25.1 mongodb() destination options 1217
8.26 mqtt(): Send messages from a local network to an MQTT broker 1232
8.26.1 Prerequisites to using the mqtt() destination 1234
8.26.2 Limitations to using the mqtt() destination 1235
8.26.3 Options of the mqtt() destination 1236
8.26.4 Possible error messages you may encounter while using the mqtt() destination 1241
8.27 network: Send messages to a remote log server using the RFC3164 protocol (network() driver) 1244
8.27.1 network() destination options 1247
8.28 Send messages to OpenObserve 1272

8.29 opensearch: Send messages to OpenSearch 1277
8.29.1 Batch mode and load balancing 1280
8.29.2 opensearch() destination options 1283
8.30 osquery: Send log messages to osquery's syslog table 1308
8.30.1 osquery() destination options 1310
8.31 Send logs, metrics, and traces to OpenTelemetry 1313
8.32 pipe: Send messages to named pipes 1337
8.32.1 pipe() destination options 1338
8.33 program: Send messages to external applications 1349
8.33.1 program() destination options 1351
8.34 pseudofile() 1366
8.34.1 pseudofile() destination options 1367
8.35 python: Write custom Python destinations 1370
8.35.1 python() destination options 1381
8.36 redis: Store name-value pairs in Redis 1395
8.36.1 Batch mode and load balancing 1397
8.36.2 redis() destination options 1398
8.37 riemann: Monitor your data with Riemann 1410
8.37.1 riemann() destination options 1412
8.38 s3: Amazon S3 1431
8.39 slack: Send alerts and notifications to a Slack channel 1442
8.39.1 Slack destination options 1445
8.40 smtp: Generate SMTP messages (emails) from logs 1465
8.40.1 smtp() destination options 1468
8.41 snmp: Send SNMP traps 1483
8.41.1 snmp() destination options 1485
8.42 splunk-hec-event: Send messages to Splunk HEC 1490
8.43 sql: Store messages in an SQL database 1493
8.43.1 Using the sql() driver with an Oracle database 1496
8.43.2 Using the sql() driver with a Microsoft SQL database 1498
8.43.3 Interacting with the database 1500
8.43.3.1 MySQL-specific interaction methods 1502
8.43.3.2 MsSQL-specific interaction methods 1503
8.43.4 sql() destination options 1504
8.44 stdout: Send messages to standard output 1523
8.45 stomp: Publish messages using STOMP 1532
8.45.1 stomp() destination options 1534
8.46 Sumo Logic destinations: sumologic-http() and sumologic-syslog() 1547
8.46.1 sumologic-http() 1551
8.46.2 sumologic-syslog() 1552
8.46.3 sumologic-http() and sumologic-syslog() destination options 1554
8.46.3.1 sumologic-http() destination options 1555

8.46.3.2 sumologic-syslog() destination options 1560
8.47 syslog: Send messages to a remote logserver using the IETF-syslog protocol 1563
8.47.1 syslog() destination options 1565
8.48 syslog-ng(): Forward logs to another syslog-ng node 1590
8.49 axosyslog-otlp(): Forward logs to another node using OpenTelemetry 1614
8.50 tcp, tcp6, udp, udp6: OBSOLETE - Send messages to a remote log server using the legacy BSD-syslog protocol (tcp(), udp() drivers) 1635
8.50.1 tcp(), tcp6(), udp(), and udp6() destination options 1636
8.50.1.1 Change an old destination driver to the network() driver 1637
8.51 telegram: Send messages to Telegram 1639
8.51.1 telegram() destination options 1641
8.52 unix-stream, unix-dgram: Send messages to UNIX domain sockets 1646
8.52.1 unix-stream() and unix-dgram() destination options 1647
8.53 usertty: Send messages to a user terminal 1663
8.54 Write your own custom destination in Java or Python 1664
8.55 Client-side failover 1665

9 log: Filter and route log messages using log paths, flags, and filters 1668

9.1 Log paths 1669
9.1.1 Embedded log statements 1672
9.1.1.1 Using embedded log statements 1674
9.1.2 if-else-elif: Conditional expressions 1677
9.1.3 Junctions and channels 1679
9.1.4 Log path flags 1683
9.2 Managing incoming and outgoing messages with flow-control 1687
9.2.1 Flow-control and multiple destinations 1693
9.2.2 Configuring flow-control 1694
9.3 Using disk-based and memory buffering 1698
9.3.1 Enabling reliable disk-based buffering 1701
9.3.2 Enabling normal disk-based buffering 1702
9.3.3 How to get information about disk-buffer files 1703
9.3.3.1 Information about disk-buffer files 1704
9.3.3.2 Getting the status information of disk-buffer files 1705
9.3.3.3 Getting the list of disk-buffer files 1707
9.3.3.4 Printing the content of disk-buffer files 1708
9.3.3.5 Orphan disk-buffer files 1709
9.3.3.6 Process orphan disk-buffer files 1711
9.3.4 Enabling memory buffering 1715
9.3.5 About disk queue files 1716
9.4 Filters 1718
9.4.1 Using filters 1719

9.4.2 Combining filters with boolean operators 1721
9.4.3 Comparing macro values in filters 1723
9.4.4 Using wildcards, special characters, and regular expressions in filters 1726
9.4.5 Tagging messages 1727
9.4.6 Filter functions 1729
9.4.6.1 facility() 1730
9.4.6.2 filter() 1732
9.4.6.3 host() 1733
9.4.6.4 in-list() 1734
9.4.6.5 level() or priority() 1736
9.4.6.6 match() 1737
9.4.6.7 message() 1738
9.4.6.8 netmask() 1739
9.4.6.9 netmask6() 1740
9.4.6.10 program() 1741
9.4.6.11 rate-limit() 1742
9.4.6.11.1 Options of rate-limit() filter 1743
9.4.6.12 source() 1744
9.4.6.13 tags() 1745
9.5 Dropping messages 1746

10 FilterX 1747

10.1 Get started with FilterX 1750
10.2 FilterX language basics 1758
10.3 Reuse FilterX blocks 1772
10.4 FilterX use cases and examples 1773
10.5 Comparing values in FilterX 1776
10.6 Boolean operators in FilterX 1780
10.7 Conditional statements in FilterX 1782
10.8 String search in FilterX 1785
10.9 IP addresses and subnets 1787
10.10 Parsing data in FilterX 1791
10.10.1 Parse CEF with FilterX 1792
10.10.2 Parse comma-separated values with FilterX 1796
10.10.2.1 Options of the parse_csv FilterX function 1800
10.10.3 Parse dates and timestamps with FilterX 1804
10.10.4 Parse JSON with FilterX 1806
10.10.5 Parse key=value pairs with FilterX 1807
10.10.5.1 Options of the parse_kv FilterX function 1809
10.10.6 Parse LEEF with FilterX 1812
10.10.7 Parse Windows Event Log XML with FilterX 1816
10.10.8 Parse XML with FilterX 1819

10.11 Format data 1821
10.11.1 Common Event Format (CEF) 1822
10.11.2 Format data as comma-separated values 1823
10.11.3 JSON 1824
10.11.4 Key-value pairs 1825
10.11.5 Log Event Extended Format (LEEF) 1827
10.11.6 RFC5424 syslog 1828
10.11.7 Windows Event Logs XML 1829
10.11.8 Format data as XML 1832
10.12 Handle and fix timezones and timestamps 1834
10.13 Handle SDATA in RFC5424 log records 1836
10.14 Handle OpenTelemetry log records 1838
10.14.1 OpenTelemetry log record fields in FilterX 1845
10.15 Create metrics with FilterX 1849
10.16 Update to FilterX 1853
10.17 FilterX operator reference 1857
10.18 FilterX function reference 1866
10.19 Troubleshoot FilterX 1908

11 Global options 1913
11.1 Configuring global options 1914
11.2 Global options reference 1915

12 TLS-encrypted message transfer 1952
12.1 Secure logging using TLS 1953
12.2 Encrypting log messages with TLS 1955
12.2.1 Configuring TLS server authentication on the AxoSyslog clients 1956
12.2.2 Configuring TLS server authentication on the AxoSyslog server 1958
12.3 Mutual authentication using TLS 1961
12.3.1 Configuring mutual TLS authentication on the AxoSyslog clients 1962
12.3.2 Configuring mutual TLS authentication on the AxoSyslog server 1965
12.4 Password-protected keys 1967
12.5 TLS options 1969

13 template and rewrite: Format, modify, and manipulate log messages 1986
13.1 Customize message format using macros and templates 1987
13.1.1 Formatting messages, filenames, directories, and tablenames 1988
13.1.2 Templates and macros 1989
13.1.3 Date-related macros 1993
13.1.4 Hard versus soft macros 1995
13.1.5 Macros of AxoSyslog 1996

13.1.6 Using template functions 2016
13.1.7 Template functions of AxoSyslog 2018
13.1.8 Modifying the on-the-wire message format 2057
13.2 Modify messages using rewrite rules 2058
13.2.1 Replace message parts 2059
13.2.2 Set message fields to specific values 2063
13.2.3 Set the severity field 2068
13.2.4 Set the facility field 2071
13.2.5 Set the priority of a message 2074
13.2.6 Set match variables 2077
13.2.7 Unset message fields 2080
13.2.8 Rename message fields 2083
13.2.9 Create custom SDATA fields 2085
13.2.10 Set multiple message fields to specific values 2087
13.2.11 map-value-pairs: Rename value-pairs to normalize logs 2090
13.2.12 Conditional rewrites 2093
13.2.13 Add and delete tags 2095
13.2.14 Rewrite the timezone of a message 2096
13.2.15 Anonymize credit card numbers 2099
13.3 Regular expressions 2101
13.3.1 Options of regular expressions 2103
13.3.2 Optimizing regular expressions 2106

14 parser: Parse and segment structured messages 2107

14.1 Apache access log parser 2108
14.1.1 Options of apache-accesslog-parser() parsers 2111
14.2 Check Point Log Exporter parser 2112
14.3 Cisco parser 2115
14.4 Parsing messages with comma-separated and similar values 2118
14.4.1 Options of CSV parsers 2122
14.5 Parsing dates and timestamps 2130
14.5.1 Options of date-parser() parsers 2132
14.6 db-parser: Process message content with a pattern database (patterndb) 2137
14.6.1 Classifying log messages 2138
14.6.1.1 The structure of the pattern database 2140
14.6.1.2 How pattern matching works 2142
14.6.1.3 Artificial ignorance 2144
14.6.2 Using pattern databases 2145
14.6.2.1 Using parser results in filters and templates 2149
14.6.2.2 Correlating log messages using pattern databases 2152
14.6.2.2.1 Referencing earlier messages of the context 2155
14.6.3 Options of db-parser() parsers 2157

14.6.4 Triggering actions for identified messages 2160
14.6.4.1 Conditional actions 2164
14.6.4.2 External actions 2166
14.6.4.3 Actions and message correlation 2168
14.6.5 Creating pattern databases 2172
14.6.5.1 Using pattern parsers 2173
14.6.5.1.1 Pattern parsers 2175
14.6.5.2 What's new in the pattern database format V5 2180
14.6.5.3 The pattern database format 2181
14.6.5.3.1 Element: patterndb 2184
14.6.5.3.2 Element: ruleset 2185
14.6.5.3.3 Element: patterns 2187
14.6.5.3.4 Element: rules 2189
14.6.5.3.5 Element: rule 2190
14.6.5.3.6 Element: patterns 2192
14.6.5.3.7 Element: urls 2194
14.6.5.3.8 Element: values 2195
14.6.5.3.9 Element: actions 2197
14.6.5.3.10 Element: action 2199
14.6.5.3.11 Element: create-context 2203
14.6.5.3.12 Element: tags 2205
14.6.5.3.13 Element: example 2206
14.6.5.3.14 Element: examples 2208
14.7 Parsing enterprise-wide message model (EWMM) messages 2209
14.8 Fortigate parser 2210
14.8.1 Fortigate parser options 2213
14.9 group-lines parser 2215
14.10 iptables parser 2223
14.11 JSON parser 2225
14.11.1 Options of JSON parsers 2229
14.12 Parsing key=value pairs 2233
14.12.1 Options of key=value parsers 2237
14.13 Linux audit parser 2240
14.13.1 Options of linux-audit-parser() parsers 2244
14.14 MariaDB parser 2247
14.15 metrics-probe 2249
14.16 Netskope parser 2257
14.17 Parse OpenTelemetry messages 2260
14.18 panos-parser(): parsing PAN-OS log messages 2263
14.18.1 Message format parsed by panos-parser() 2265
14.18.2 PAN-OS parser options 2266
14.19 PostgreSQL csvlog 2267

14.20 Python parser 2270
14.21 Regular expression (regexp) parser 2281
14.21.1 Options of Regular expression parsers 2284
14.22 Structured data (SDATA) parser 2287
14.23 Sudo parser 2289
14.24 Parsing syslog messages 2291
14.24.1 Options of syslog-parser() parsers 2295
14.25 Parsing tags 2304
14.26 Websense parser 2305
14.27 Windows XML Event Log (EVTX) parser 2308
14.28 XML parser 2309
14.28.1 Limitations of the XML parsers 2314
14.28.2 Options of the XML parsers 2316

15 Correlating log messages 2320

15.1 Correlating messages using the grouping-by() parser 2321
15.1.1 Referencing earlier messages of the context 2326
15.1.2 Options of grouping-by parsers 2328

16 Enriching log messages with external data 2334

16.1 Adding metadata from an external file 2335
16.1.1 Using filters as selector 2338
16.1.2 Shell-style globbing in the selector 2340
16.1.3 Options add-contextual-data() 2342
16.2 Looking up GeoIP data from IP addresses (DEPRECATED) 2345
16.2.1 Options of geoip parsers 2349
16.3 Looking up GeoIP2 data from IP addresses 2350
16.3.1 Referring to parts of the message as a macro 2351
16.3.2 Using the GeoIP2 parser 2352
16.3.3 Transferring your logs to Elasticsearch using GeoIP2 2355
16.3.4 Options of geoip2 parsers 2356

17 Statistics and metrics of AxoSyslog 2358

17.1 Collect metrics with Prometheus 2359
17.2 Metrics reference 2362
17.3 Statistics reference 2372
17.4 Log statistics from the internal() source 2379

18 Multithreading and scaling 2381

18.1 Multithreading concepts 2382
18.2 Configuring multithreading 2385
18.3 Optimizing multithreaded performance 2386

19 Nonsequential message processing 2387

20 Writing Python modules 2389

21 Best practices and examples 2406

21.1 General recommendations 2407
21.2 Handling large message load 2408
21.3 Using name resolution in syslog-ng 2409
21.3.1 Resolving hostnames locally 2410
21.4 Collecting logs from chroot 2411
21.5 Configuring log rotation 2412
21.6 Load balancing logs between multiple destinations 2414
21.6.1 Load balancing with a round robin load balancing method based on the R_MSEC macro 2415
21.6.2 Configuration generator for the load balancing method based on MSEC hashing 2417

22 Troubleshooting 2418

22.1 Possible causes of losing log messages 2420
22.2 Creating core files 2422
22.3 Collecting debugging information with strace, truss, or tusc 2423
22.4 Running a failure script 2424
22.5 Stopping the syslog-ng process 2426
22.6 Reporting bugs and finding help 2427
22.7 Recover data from orphaned diskbuffer files 2428
22.8 Error messages 2429
22.9 Error: unexpected LL_IDENTIFIER 2432
22.10 SELinux prevents using the execmem access on a process 2434

23 Getting support 2435

24 Manual pages 2436

24.1 The syslog-ng manual page 2437
24.2 The syslog-ng.conf manual page 2443
24.3 The syslog-ng-ctl manual page 2453
24.4 The dqtool tool manual page 2471
24.5 The loggen manual page 2477
24.6 The pdbtool manual page 2484
24.7 The persist-tool manual page 2494
24.8 The secure-logging manual page 2498
24.9 The slogencrypt manual page 2508
24.10 The slogkey manual page 2511
24.11 The slogverify manual page 2514
24.12 The syslog-debun manual page 2517

25 The syslog-ng OSE Documentation License 2523

1 What's new

New features and changes in each AxoSyslog release.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

Before you upgrade, check the deprecations and breaking changes of every version between your current version and the target version. For the upgrade steps, see Upgrade AxoSyslog to a newer version.

Version 4.29 (2026-10-07)

- You can now change the `capacity-bytes()` option of an existing disk-buffer file. The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- The new `syslog-ng --list-paths` command-line option prints the files and directories that the configuration refers to, without starting AxoSyslog.
- The `azure(monitor())` authentication of `cloud-auth()` has a new `auth-url()` option to set the login host that the access token is requested from. Previously, the token endpoint was always `https://login.microsoftonline.com` , so only the public Azure cloud was available.
- When batching is enabled in the `http()` destination, templated `headers()` now require the `worker-partition-key()` option, the same way as templated `url()` and `body-prefix()` options do. Previously, the header values of the first message were used for the whole batch. In addition, `worker-partition-key()` is now required only if the templates actually depend on the message.

For a list of bugfixes, see the GitHub release page .

Version 4.28 (2026-09-16)

- The new `elasticsearch-bulk()` source implements the Elasticsearch Bulk API, so Elastic Agent, Beats, and other clients can send their events to AxoSyslog by pointing their Elasticsearch output at it.
- The new `splunk-hec()` source receives messages from Splunk HTTP Event Collector (HEC) clients, so you can point your existing HEC clients at AxoSyslog and process or route the events before forwarding them.
- The new `ehttp()` source receives log messages over HTTP and HTTPS. Unlike the Python-based `webhook()` source , it is implemented natively in AxoSyslog, and it is intended to replace `webhook()` in the long run. Note that `ehttp()` is experimental: it will be renamed to `http()` once its options are considered stable, and its options can change until then.
- The `opentelemetry()` source now supports the `mode()` option. With `mode(filterx-dict)` , the source converts incoming log records directly into the declared `log` , `resource` , and `scope` FilterX variables as plain dictionaries, instead of creating the `${.otel_raw.*}` name-value pairs. This removes an extra serialization step, so processing OpenTelemetry logs in FilterX becomes significantly faster. The default remains `mode(logmessage)` , which keeps the previous behavior.

  The `opentelemetry()` source now also sets the `.tls.x509_cn` , `.tls.x509_o` and `.tls.x509_ou` name-value pairs from the client certificate, the same way the `network()` and `syslog()` sources do it.
- The `format_kv` FilterX function now supports the `quote_char` and `always_quote` options to control how values are quoted.
- The `update_metric` FilterX function now supports the `set` option, which assigns an absolute value to the counter instead of incrementing it. This makes `update_metric` usable for gauge-like metrics.

For a list of bugfixes, see the GitHub release page .

Version 4.27 (2026-08-19)

- FilterX has a new `tuple` variable type, a read-only, list-like data type similar to a Python tuple. You can initialize a tuple only once, after that it remains read-only until the end of its lifecycle.

- The new `uuid7` FilterX function generates a random RFC 9562 UUIDv7 identifier. Such identifiers embed a millisecond-precision Unix timestamp, so they sort lexically by creation time. In addition, the `uuid` function is now also available under the `uuid4` alias.
- The `http()` , `splunk-hec-event()` , and `openobserve()` destinations now support the `response-adapter()` option. Some servers put the error data in the HTTP response body instead of the status code, and this option lets AxoSyslog find these errors and retry the affected batch.
- In the `arrow-flight()` destination, the `timeout()` option is deprecated. Use the `keep-alive()` option instead.

For a list of bugfixes, see the GitHub release page .

Version 4.26 (2026-06-30)

- The new `arrow-flight()` destination sends structured, columnar data to an Apache Arrow Flight server over gRPC. You define the columns and their types with the `schema()` option and map each column to a AxoSyslog template.
- The `network()` and `syslog()` sources and destinations now support the `trusted-fingerprints()` TLS option to accept connections only from hosts presenting a certificate with a listed fingerprint. Unlike the now-deprecated `trusted-keys()` option, which was limited to SHA-1, `trusted-fingerprints()` accepts fingerprints computed with any OpenSSL-supported digest algorithm, such as SHA-256 or SHA-512.
- The FilterX `switch` statement now supports ranged case matching for integer targets, for example, `case 1..4:` .
- The `usertty()` destination now supports the `escaping()` option, which turns on escaping for the `'` , `"` , and backslash characters. This is useful when the messages are passed to an application that cannot handle escaped characters properly.
- AxoSyslog packages are now available for RHEL 10 and AlmaLinux 10 , Fedora 44 , and Ubuntu 26.04 .

For a list of bugfixes, see the GitHub release page .

Version 4.25 (2026-05-05)

- The `cache_json_file` FilterX function now accepts an optional `default_value` parameter. When the referenced JSON file is missing or fails to load, AxoSyslog uses `default_value` instead of aborting with a configuration error.

- New FilterX functions for common data transformations:
  - Cryptographic digests: `md5` , `sha1` , `sha256` , and `sha512` return the hash of a string or bytes value as a lowercase hexadecimal string. The generic `digest` function returns the raw hash as bytes and accepts an optional `alg=` parameter to select the algorithm.
  - Encoding and decoding: `base64_encode` / `base64_decode` , `urlencode` / `urldecode` , and `hex_encode` / `hex_decode` .
  - UTF-8 handling: `utf8_validate` checks whether a string is valid UTF-8, and `utf8_sanitize` replaces invalid byte sequences with their `\xNN` escaped representation.
- The `glob_match` FilterX function matches a string against one or more glob patterns.
- FilterX has two new variable types to work with network addresses: `ip()` represents a single IPv4 or IPv6 address, while `subnet()` represents an IPv4 or IPv6 subnet in CIDR notation. You can use these types together with the list membership operator (`in`) to check whether an address belongs to a subnet, which is useful for filtering messages based on the source or destination network of the traffic they describe. For details, see IP addresses and subnets.
- The `uuid` FilterX function generates a random UUID v4 string.
- The `network()` and `syslog()` sources now support `transport("proxied-udp")` to receive UDP syslog messages from load balancers that use HAProxy Proxy Protocol v2 to preserve original client IP addresses.

For a list of bugfixes, see the GitHub release page .

Version 4.24 (2026-03-23)

- You can query the list of supported metrics by running `syslog-ng --metrics-registry` . For a reference of available metrics, see Metrics reference.
- The new `batch-idle-timeout()` option closes the batch if no new message is received since the last message within the `batch-idle-timeout()` period. The following destinations support `batch-idle-timeout()` : `amqp()` , `bigquery()` , `clickhouse()` , `discord()` , `elasticsearch()` , `mongodb()` , `google-pubsub()` , `http()` , `kafka-c()` , `opensearch()` , `opentelemetry()` , `redis()` , `riemann()` , `slack()` , `smtp()` , `stomp()` , `sql()` , `sumologic()` .
- New FilterX functions to manipulate the timezone of the messages: Handle and fix timezones and timestamps.
- `network-load-balancer()` now supports the `failover()` option.

For a list of bugfixes, see the GitHub release page .

Version 4.23 (2026-02-18)

- The `opentelemetry()` source now supports the `ip()` option to specify the bind address.
- New FilterX functions:
  - `format_isodate` formats a date as ISODATE: `%Y-%m-%dT%H:%M:%S%z`
  - `move` moves the specified variable to its new location, instead of copying it.
- The `http()` destination option `content-compression()` sends the payload compressed only if the compressed content is smaller than the original plain-text content. This can cause problems if your destination server accepts only compressed payload. Setting `force-content-compression(yes)` instructs AxoSyslog to always send the compressed payload.
- The `network()` and `syslog()` sources and destinations now support the `extended-key-usage-verify()` option to check that the certificate of the peer contains the appropriate value in its Extended Key Usage (EKU) field. (Clients should have Client Auth EKU values (`X509_PURPOSE_SSL_CLIENT`), while server should have Web Server Auth (`X509_PURPOSE_SSL_SERVER`).)
- The amount of memory AxoSyslog uses to store queued messages has been significantly reduced, by up to 50%.

For a list of bugfixes, see the GitHub release page .

Version 4.22 (2026-01-21)

The FilterX list membership operators now support searching the keys of dictionaries.

For a list of bugfixes, see the GitHub release page .

Version 4.21 (2025-12-15)

- A new FilterX function `format_syslog_5424` that formats messages as an RFC5424 (IETF-syslog) syslog message.
- Syslog sources can now set `transport(nul-terminated)` to receive messages that use NUL characters instead of newline characters to separate log records.

- Destinations that support worker partitioning (the `worker-partition-key()` option) can now set `worker-partition-autoscaling(yes)` to allow each worker to maximize its batch size.

For a list of bugfixes, see the GitHub release page .

Version 4.20 (2025-11-20)

- The `clickhouse()` destination now has a `format()` option, allowing you to send data in a compact `JSONCompactEachRow` format.
- The `opentelemetry()` and `axosyslog-otlp()` (formerly called `syslog-ng-otlp()`) sources now support the `keep-alive()` option, and is enabled by default.
- The `s3()` destination has a new option:
  - You can now set a suffix for your S3 objects using the `object-key-suffix()` option .
- The `http()` and other threaded destinations now have a `worker-partition-buckets()` option that determines the number of worker threads used for the `worker-partition-key()` .

For a list of bugfixes, see the GitHub release page .

Version 4.19 (2025-10-15)

- The `dict_to_pairs` FilterX function can convert a dictionary to a list of pairs.
- The `includes` FilterX function can limit the length of the search string.

Version 4.18 (2025-09-30)

- You can now use macros and templates in the `headers()` option of the `http()` destination to set the headers dynamically.
- The `parse_csv` FilterX parser now supports the `quote-pairs` option.
- `+` and `-` unary operators for FilterX.

Version 4.17 (2025-09-04)

- The `parse_kv` FilterX function has an option (stray_words_key) to append stray words to the preceding key.

- You can now use negative indexes when slicing FilterX strings .
- The `dpath` FilterX function assigns a value to a dictionary and creates any elements of the path that don't exist.
- When using `parallelize()` during Nonsequential message processing, you set the `batch-size()` option to specify how many consecutive messages should be processed by a single `parallelize()` worker.
- For the `clickhouse()` destination, you can now use the `json-var()` option to send the message to the ClickHouse server in Protobuf/JSON mixed mode (`JSONEachRow` format). In this mode, type validation is performed by the ClickHouse server itself, so no Protobuf schema is required for communication.

Version 4.16 (2025-08-15)

- New `${PROTO_NAME` macro .
- New FilterX functions `str_strip` , `str_lstrip` , `str_rstrip` to remove the leading and/or trailing whitespaces from a string.
- The `batch-timeout()` option of the following destinations now defaults to `0` : `google-pubsub()` , 'logscale()' , `openobserve()` , `splunk()` .

Breaking change

The name of some fields changed in the `parse_cef` and `parse_leef` parsers to avoid name collisions with fields in the extensions:

For CEF:

- `version` -> `cef_version`
- `name` -> `event_name`

For LEEF:

- `version` -> `leef_version`
- `vendor` -> `vendor_name`
- `delimiter` -> `leef_delimiter`

Version 4.15 (2025-08-01)

- You can exclude files in the `wildcard-file()` source using the `exclude-pattern()` option.

- You can use templates in the `body-prefix()` option of the `http()` destination , and in destinations based on `http()` .
- ADC authentication now can use `service-account-key()` .
- `gcp(service-account())` authentication can now use `scope()` instead of `audience()` .
- New FilterX features:
  - `str_replace` function for string replacement.
  - String slicing operator (`..`) .
  - Create dict element if non-null (`:??`) operator.

Version 4.14 (2025-07-18)

- New `client-port` option for `loggen` .
- Starting with version 4.14, running `syslog-ng-ctl stats` automatically shows orphan counters to avoid losing information. Information loss could happen, for example, when sending messages using short-lived (few seconds long) connections, while scraping metrics in minute intervals.
- The `cisco-parser()` now handles the Cisco Nexus NXOS 9.3 syslog format.

Version 4.13 (2025-07-08)

- You can format arbitrary data as protobuf using the specified schema (proto file) using the protobuf_message FilterX function. Also, you can send such pre-formatted data using the `proto-var()` option of the ClickHouse and Google BigQuery destinations.
- You can now format dictionaries as XML and Windows Event Log XML using the `format_xml` and `format_windows_eventlog_xml` FilterX functions.
- You can now format dictionaries as CEF and LEEF messages using the `format_cef` and `format_leef` FilterX functions.
- Changes in the `parse_cef` and `parse_leef` FilterX functions:

separate_extensions

Available in AxoSyslog 4.13 and later.

Starting with version 4.13, AxoSyslog parses fields from extensions to the same level as regular fields. In earlier versions, extensions were grouped under the `extensions` key. To keep using the `extensions` key, set `separate_extensions=true`.

Default value: `false`

Version 4.12 (2025-06-18)

- Starting from this version, AxoSyslog is licensed under GPL-3.0-or-later. For details, see Product licensing and the AxoSyslog License Update: Moving to GPL3 blog post.
- New FilterX features:
  - Arithmetic operators : `+` (addition), `-` (substraction), `*` (multiplication), `/` (division), and `%` (modulo)
  - List membership operator (`in`) : checks if a value is present in a list.
  - `strcasecmp` function for case insensitive string comparison.
  - To help troubleshooting FilterX blocks , AxoSyslog now includes specific functions that allow you to track failures in FilterX code.
- Way to propagate the type information of the data fields in the `clickhouse()` destination using the ClickHouse format schema. For details, see server-side-schema().
- You can enable flow-control for every log path using the `log-flow-control()` global option .
- `loggen` has a new `--perf` option to disable rate limiting.
- Internal STATS messages are now disabled by default .

Version 4.11 (2025-04-09)

- Documentation for the Webhook.
- New macros: PEERIP and PEERPORT
- New gRPC option `response-action()` for the `bigquery()` , `clickhouse()` , `google-pubsub-grpc()` , `loki()` , and the `opentelemetry()` destinations.
- New FilterX functions: set_pri, set_timestamp.
- Updates the Azure Monitor destination .

Version 4.10 (2025-02-13)

- New Google Pub/Sub gRPC destination to send logs and data to Google Pub/Sub via gRPC.

- New destination to send logs and data to Azure Monitor and Microsoft Sentinel . For details, see Send data to Azure Monitor and Sentinel.
- New `$SOURCEPORT` macro which expands to the source port of the peer.
- The `syslog()` source driver can now auto-detect RFC6587-style octet-count based framing.
- Lots of FilterX updates, including a new `=??` operator and several functions:
  - keys
  - load_vars
  - metrics_labels
  - set_fields
  - strftime
- Switch-case expressions in FilterX to better organize the code instead of using multiple `if` , `elif` , `else` blocks. Using switch-case expressions also improves performance.

Version 4.9 (2024-11-11)

- ClickHouse database destination.
- Log tapping with the `syslog-ng-ctl attach` command .
- FilterX data parsing and processing engine.
- Updated lists of available options for the gRPC-based destinations (`bigquery()` , `loki()` , `opentelemetry()` , `syslog-ng-otlp()`). You can now also set dynamic header values for these destinations.
- Added the `idle-timeout()` option to file() source options, stdin() source options, systemd-syslog() source options, wildcard-file() source options, pipe() source options, program() source options, unix-stream() and unix-dgram() source options.

  These sources have a new `exit-on-eof` flag that makes AxoSyslog stop when EOF is received.
- Added the MSGFORMAT macro.
- Added `.tls.x509_fp` to .tls.x509.

Other documentation updates

- Cloud authentication option updates for the `http()` and `google-pubsub` destinations.

- `syslog-ng-ctl list-files` command lists files referenced in your configuration, for example, certificates or external configuration files.
- `lifetime()` global option to prune dynamic counters.

Version 4.8 (2024-07-12)

- APT repository for Debian and Ubuntu based systems.
- You can send messages and metrics to Elasticsearch data streams to store your log and metrics data as time series data using the `elasticsearch-datastream()` destination driver.
- You can use the `server-side-encryption()` and `kms-key()` options to configure encryption for Amazon S3 destinations .
- You can now set static gRPC headers in the `bigquery()` , `loki()` , and the `opentelemetry()` destinations.
- The `opentelemetry()` parser has a new `set-hostname()` option.

Version 4.7 (2024-04-18)

- Arr logs source
- Jellyfin logs source
- `channel-args()` option for gRPC-based drivers, like `opentelemetry()`
- `concurrent-requests()` option for the `opentelemetry()` source and the `syslog-ng-otlp()` source
- tenant-id() option for the Loki destination
- tags-head template function
- MQTT_TOPIC macro
- TRANSPORT macro updates

For details, see the release announcement blog post .

Version 4.6 (2024-02-01)

- Google BigQuery destination
- Windows XML Event Log (EVTX) parser
- tag template function
- `batch-bytes()` , `compression()` and `workers()` options for the syslog-ng-otlp() and opentelemetry() destinations

For details, see the release announcement blog post .

New sources

- New macOS sources
- qBittorrent logs
- Pi-hole Faster Than Light logs

2023-10-20 to version 4.5 release (2024-01-05)

- Google Pub/Sub destination
- OpenObserve destination
- New `http()` destination options Templates in the url() and worker-partition-key()

Parsers

- New PostgreSQL csvlog parser
- Columnless mode in csv-parser

TLS options

- ssl-version()
- `ignore-validity-period` option in to ssl-options()

Manual pages

- `--check-startup` in The syslog-ng manual page
- secure-logging , slogencrypt , slogkey , and slogverify manual pages.

Other changes

- New quickstart section Sending Kubernetes logs to OpenSearch
- Updates in Install AxoSyslog with Podman and Install AxoSyslog with Docker
- `close_batch` and `set_transport` methods in the python source

2023-08-18 to 2023-10-20

- syslog-ng-otlp source and syslog-ng-otlp destination
- Loki destination
- Amazon S3 destination
- OpenSearch destination
- stdout destination
- `http` destination options:
  - ignore-hostname-mismatch ssl-option
  - accept-encoding()
  - accept-redirects()
  - content-compression()
- Dynamic labeling in the metrics-probe parser

2023-07-07 to 2023-08-18

OpenTelemetry

- OpenTelemetry source
- OpenTelemetry destination
- OpenTelemetry parser

New sources and related changes

- Hypr Audit Trail and Hypr App Audit Trail
- OpenTelemetry
- `match-boot` and `matches` options of the `syslog-journal()` source

New destinations and related changes

- Falcon LogScale
- OpenTelemetry
- Splunk HEC
- `dbdi-driver-dir()` and `quote-char()` options of the `sql()` destination
- `bulk-mode()`, `bulk-bypass-validation()`, and `bulk-unordered()` and `write-concern()` options of the `mongodb()` destination

New parsers and related changes

- group-lines
- Count the messages that pass through the log path (metrics-probe)
- OpenTelemetry
- RFC5424 structured data (SDATA) parser
- `sdata-prefix` option for the `syslog-parser`
- `escape-backslash-with-sequences` option for the `csv-parser`

Other changes

- Typing support (Specifying data types in value-pairs)
- Nonsequential message processing for improved performance
- An overview of writing Python modules for syslog-ng
- New syslog-ng-ctl commands
- Configuration identifier
- Named log paths
- format-date template function
- TLS improvements: OCSP stapling verification and SSL_CONF_cmd support
- RAWMSG_SIZE macro
- `lower` and `upper` transformations for the rekey `value-pairs()` option

New global options

- log-level
- stats

2 Introduction to AxoSyslog

What AxoSyslog is and is not, what you can use it for, who uses it, and which platforms it supports.

What AxoSyslog is

AxoSyslog is a scalable, open source security data processor by Axoflow .

AxoSyslog is a drop-in replacement for `syslog-ng`, created by the original creators of `syslog-ng`. (It started as a fork, branched after syslog-ng™ v4.7.1).

AxoSyslog, the drop-in `syslog-ng` replacement

AxoSyslog provides the:

- same binaries (for example, `/usr/sbin/syslog-ng`),
- same configuration files (`/etc/syslog-ng/syslog-ng.conf`), certificates, etc.,
- same configuration syntax, and the
- same license and development practices.

If you're already using `syslog-ng` , you can upgrade your existing `syslog-ng` deployments to AxoSyslog in a matter of minutes.

More than just a replacement

AxoSyslog also provides:

- detailed metrics about what your pipeline is doing,
- the FilterX data processing engine ,
- enhanced container and Kubernetes support, including Helm charts ,
- eBPF integration for efficient kernel-level data collection with minimal overhead,
- destinations for ClickHouse , Microsoft Sentinel, Azure Monitor , Google Pub/Sub , and many more modern services,
- regular releases with new features, and
- responsive and helpful community and professional support for AxoSyslog and `syslog-ng` .

The AxoSyslog application is a flexible and highly scalable data processing, observability, and system logging application that is ideal for creating centralized and trusted logging and observability solutions and telemetry pipelines, supporting on-premises, cloud-native, and hybrid environments.

Unlike a traditional syslog daemon, AxoSyslog is actively developed with frequent releases that add modern protocols, integrations, and processing features – see What's new.

Among others, AxoSyslog allows you the following.

Process and transform data with FilterX

AxoSyslog includes FilterX , a powerful data processing engine for parsing, filtering, enriching, and reshaping log and observability data with a concise, expressive language. FilterX makes it easy to work with structured data formats like JSON, and to build sophisticated pipelines that go far beyond traditional syslog processing.

Collect and forward data using OpenTelemetry and modern protocols

AxoSyslog natively supports the OpenTelemetry protocol (OTLP) to receive and forward logs, metrics, and traces, so you can integrate AxoSyslog into modern observability and telemetry pipelines. To move data efficiently between AxoSyslog nodes, use the `axosyslog-otlp()` drivers . AxoSyslog also fully supports the traditional syslog protocols – legacy BSD syslog (RFC3164) and the enhanced RFC5424 – as well as delivering data over HTTP in JavaScript Object Notation (JSON) to popular SIEMs and analytics engines.

Build observability and telemetry pipelines

AxoSyslog is a building block for end-to-end observability pipelines that exposes detailed metrics about its own operation, so you can monitor the health and throughput of your pipeline.

Run natively in cloud-native and Kubernetes environments

The AxoSyslog application is the ideal choice to collect logs and observability data in massively heterogeneous environments that use several different operating systems, and it runs natively in containerized and Kubernetes environments. You can deploy it using the official Helm charts .

Flexible data extraction and processing

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

Filter and classify

The AxoSyslog 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 logical operators offers almost unlimited flexibility to forward only the important data to the selected destinations.

Parse and rewrite

The AxoSyslog application can segment the incoming data 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 data, AxoSyslog allows you to correlate multiple log messages and aggregate the extracted information into a single message. You can also use external information to enrich your log data .

Store data in big-data and cloud destinations

The data that your organization has to process, store, and review increases daily, so many organizations use big data and cloud solutions for their logs. To accommodate this huge amount of data, AxoSyslog natively supports storing log messages and observability data in destinations like Elasticsearch /OpenSearch /OpenObserve , modern analytics services like ClickHouse and Microsoft Sentinel and Azure Monitor , and cloud storage solutions like Google BigQuery , Google Pub/Sub and Amazon S3 .

Message queue support

Large organizations increasingly rely on queuing infrastructure to transfer their data. For that purpose, AxoSyslog supports Apache Kafka , the Advanced Message Queuing Protocol (AMQP) , MQTT , 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 AxoSyslog application supports the following databases: MongoDB , MSSQL, MySQL, Oracle, PostgreSQL, and SQLite .

AxoSyslog also allows you to extract the information you need from your data, and directly send it to your Graphite , Redis , or Riemann monitoring system.

Secure and reliable data transfer

The AxoSyslog application enables you to send log messages and other observability data from your hosts to remote servers using the latest protocol standards. You can collect and store your data centrally on dedicated solutions, like log servers or SIEMs. Transfer your log messages and observability data using traditional and modern, reliable protocols to ensure that no messages are lost.

Disk-based message buffering

To minimize the risk of losing important log messages, the AxoSyslog application can store messages on the local hard disk if the central log server or the network connection becomes unavailable. The AxoSyslog 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 AxoSyslog is restarted.

Secure logging using TLS

Log messages may contain sensitive information that should not be accessed by third parties. Therefore, AxoSyslog 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.

IPv4 and IPv6 support

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

What AxoSyslog is not

The AxoSyslog 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 AxoSyslog cannot interpret and analyze the meaning behind the messages, or recognize patterns in the occurrence of different messages.

Why do you need AxoSyslog

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 AxoSyslog 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.

Beyond traditional centralized logging, AxoSyslog acts as the collection and processing layer of a modern observability and security pipeline. It ingests logs, metrics, and traces, processes and enriches them with FilterX , and routes them – over the OpenTelemetry protocol , HTTP, or traditional syslog – to both legacy stores and modern backends such as OpenSearch, ClickHouse, Microsoft Sentinel, and cloud storage. This lets you feed your SIEM, observability platform, and long-term archive from a single, vendor-neutral pipeline.

Who uses AxoSyslog

The AxoSyslog 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 AxoSyslog is particularly advantageous for:

- Platform engineering and site reliability (SRE) teams running containerized and Kubernetes workloads
- Security and SOC teams that feed logs into a SIEM
- Observability teams building telemetry pipelines for logs, metrics, and traces
- Cloud-native organizations operating in hybrid and multi-cloud environments

- 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.

3 Quick-start guide

Configure AxoSyslog as a client, server, or relay, manage the service on Linux, and send Kubernetes logs to OpenSearch.

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

AxoSyslog can transport log messages between nodes using several protocols. For transport between AxoSyslog nodes, you can use the legacy BSD-syslog and the IETF-syslog protocols, or the OpenTelemetry protocol with the `axosyslog-otlp()` drivers. The client and server pages below show how to configure both. For details about the `axosyslog-otlp()` drivers, see axosyslog-otlp(): Forward logs to another node using OpenTelemetry and axosyslog-otlp(): Receive logs from another node using OpenTelemetry.

- To configure AxoSyslog as a client that sends log messages to a central logserver, see Configuring AxoSyslog on client hosts.
- To configure AxoSyslog as a server that receives log messages from client hosts, see Configuring AxoSyslog as a logserver.
- To configure AxoSyslog as a relay that receives log messages from client hosts and forwards them to a central logserver, see Configuring AxoSyslog relays.
- For information about managing and checking AxoSyslog services on Linux, see Managing and checking the syslog-ng service on Linux.

Next steps

To filter, parse, and modify your log messages, see Get started with FilterX.

3.1 Configuring AxoSyslog on client hosts

To configure AxoSyslog on a client host, complete the following steps. For an overview of how clients fit into a logging infrastructure, see Client mode.

1. Install the AxoSyslog application on the host. For details installing AxoSyslog on specific operating systems, see Install AxoSyslog.

2. Configure the local sources to collect the log messages of the host. Starting with version 3.2, AxoSyslog 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: Collect the system-specific log messages of a platform.

   To configure AxoSyslog, edit the `syslog-ng.conf` file with any regular text editor application. The location of the configuration file depends on the platform you are running AxoSyslog, and how you have installed AxoSyslog it.

   - **Native packages** of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` directory.
   - **Containers**: When running AxoSyslog in a container, typically you map an external file to the `/etc/syslog-ng/syslog-ng.conf` file within the container. Check the mapped volumes of your container, and edit the external file.
   - **Kubernetes**: If you're running AxoSyslog in Kubernetes and have installed it with helm, usually you configure AxoSyslog by editing a `values.yaml` file, and redeploying AxoSyslog. Often the `syslog-ng.conf` part is under the `config.raw` section in the `values.yaml` file. For details, see Parameters of the AxoSyslog Helm chart.

   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: Collect messages from text files.

> **Note**
>
> **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 AxoSyslog collects platform-specific log messages and the internal log messages of AxoSyslog.

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

3. Create a network destination that points directly to the AxoSyslog 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"));
};
```

When both nodes run AxoSyslog, you can use the OpenTelemetry protocol with the `axosyslog-otlp()` destination instead. It preserves the internal representation of the messages, and adds scalability (via the `workers()` option), application-layer acknowledgement, and improved load balancing:

```
destination d_otlp {
    axosyslog-otlp(url("10.1.2.3:4317"));
};
```

> **Note**
>
> **The `axosyslog-otlp()` driver is available in AxoSyslog version 4.12 and later, and requires the `axosyslog-grpc` (or `axosyslog-mod-grpc`) package. For all options, see axosyslog-otlp(): Forward logs to another node using OpenTelemetry. To send logs to a third-party OpenTelemetry collector or backend instead of another AxoSyslog node, use the Send logs, metrics, and traces to OpenTelemetry destination.**

4. Create a log statement connecting the local sources to the AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog

The following is the default configuration file of AxoSyslog. It collects local log messages and the log messages of AxoSyslog and saves them in the `/var/log/messages` file.

```
@version: 4.29
@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: 4.29
@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);
};
```

Example: An OpenTelemetry client configuration

The following is a simple configuration file that collects local log messages and forwards them to another AxoSyslog node using the OpenTelemetry protocol.

```
@version: 4.29
@include "scl.conf"
source s_local {
    system(); internal();
};
destination d_otlp {
    axosyslog-otlp(url("192.168.1.1:4317"));
};
log {
    source(s_local); destination(d_otlp);
};
```

3.2 Configuring AxoSyslog as a logserver

To configure AxoSyslog on a server host, complete the following steps. For an overview of how servers fit into a logging infrastructure, see Server mode.

1. Install the AxoSyslog application on the host. For details installing AxoSyslog on specific operating systems, see Install AxoSyslog.

2. AxoSyslog 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: Collect the system-specific log messages of a platform.

3. To configure AxoSyslog, edit the `syslog-ng.conf` file with any regular text editor application. The location of the configuration file depends on the platform you are running AxoSyslog, and how you have installed AxoSyslog it.

   - **Native packages** of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` directory.
   - **Containers**: When running AxoSyslog in a container, typically you map an external file to the `/etc/syslog-ng/syslog-ng.conf` file within the container. Check the mapped volumes of your container, and edit the external file.
   - **Kubernetes**: If you're running AxoSyslog in Kubernetes and have installed it with helm, usually you configure AxoSyslog by editing a `values.yaml` file, and redeploying AxoSyslog. Often the `syslog-ng.conf` part is under the `config.raw` section in the `values.yaml` file. For details, see Parameters of the AxoSyslog Helm chart.

   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"));
};
```

If your clients are also AxoSyslog nodes, you can receive their messages over the OpenTelemetry protocol with the `axosyslog-otlp()` source. This preserves the internal representation of the messages, and adds scalability, application-layer acknowledgement, and improved load balancing:

```
source s_otlp {
    axosyslog-otlp(port(4317));
};
```

> **Note**
>
> **The `axosyslog-otlp()` driver is available in AxoSyslog version 4.12 and later, and requires the `axosyslog-grpc` (or `axosyslog-mod-grpc`) package. For all options, see axosyslog-otlp(): Receive logs from another node using OpenTelemetry. To receive data from a third-party OpenTelemetry sender rather than another AxoSyslog node, use the Receive logs, metrics, and traces from OpenTelemetry source.**

For other options, see syslog: Collect messages using the IETF-syslog protocol and tcp, tcp6, udp, udp6: OBSOLETE - Collect messages from remote hosts using the BSD syslog protocol.

> **Note**
>
> **Starting with AxoSyslog 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 AxoSyslog 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 AxoSyslog server treats 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 AxoSyslog server, use the `keep-hostname(yes)` option both on the AxoSyslog relay and the AxoSyslog 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 AxoSyslog that collects incoming log messages and stores them in a text file.

```
@version: 4.29
@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);
};
```

Example: An OpenTelemetry server configuration

The following configuration receives log messages from AxoSyslog clients over the OpenTelemetry protocol and stores them in a text file.

```
@version: 4.29
@include "scl.conf"
source s_local {
    system(); internal();
};
source s_otlp {
    axosyslog-otlp(port(4317));
};
destination d_logs {
    file("/var/log/syslog-ng/logs.txt");
};
log {
    source(s_local); source(s_otlp); destination(d_logs);
};
```

3.3 Configuring AxoSyslog relays

This section describes how to configure AxoSyslog as a relay. For an overview of relays and their typical use cases, see Relay mode.

3.3.1 Configuring AxoSyslog on relay hosts

To configure AxoSyslog on a relay host, complete the following steps:

1. Install the AxoSyslog application on the host. For details on installing AxoSyslog on specific operating systems, see Install AxoSyslog.
2. Configure the network sources that collect the log messages sent by the clients.
3. Create a network destination that points to the AxoSyslog server.

> **Note**
>
> **If both your clients (or the relay) and the upstream server run AxoSyslog, you can use the OpenTelemetry protocol on both legs instead of the syslog protocol: receive messages with an `axosyslog-otlp()` source and forward them with an `axosyslog-otlp()` destination. The `axosyslog-otlp()` driver is available in AxoSyslog version 4.12 and later, and requires the `axosyslog-grpc` (or `axosyslog-mod-grpc`) package. For details, see axosyslog-otlp(): Receive logs from another node using OpenTelemetry and axosyslog-otlp(): Forward logs to another node using OpenTelemetry.**

4. Create a log statement connecting the network sources to the AxoSyslog 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 AxoSyslog server.
7. Enable the `keep-hostname()` and disable the `chain-hostnames()` options. (For details on how these options work, see Global options .)

> **Note**
>
> **It is recommended to use these options on your AxoSyslog server as well.**

8. Set filters and options (for example, TLS encryption) as necessary.

> **Note**
>
> By default, the AxoSyslog server treats 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 AxoSyslog server, use the `keep-hostname(yes)` option both on the AxoSyslog relay and the AxoSyslog 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: 4.29
@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);
};
```

Example: A relay configuration using OpenTelemetry

The following relay configuration receives log messages from AxoSyslog clients over the OpenTelemetry protocol and forwards them to the upstream AxoSyslog server using the same protocol.

```
@version: 4.29
@include "scl.conf"
options {
    time-reap(30);
    mark-freq(10);
    keep-hostname(yes);
    chain-hostnames(no);
};
source s_local {
    system(); internal();
};
source s_network {
    axosyslog-otlp(port(4317));
};
destination d_otlp {
    axosyslog-otlp(url("192.168.1.5:4317"));
};
log {
    source(s_local); source(s_network);
    destination(d_otlp);
};
```

3.3.2 How relaying log messages works

Depending on your exact needs about relaying log messages, there are many scenarios and AxoSyslog 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 > relay > server*, where the IP address of `client-host` is `192.168.1.2`. The `client-host` device sends a syslog message to `relay`. Depending on the settings of `relay`, the following can happen.

- By default, the `keep-hostname()` option is disabled, so `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 `relay`, but name resolution is disabled (the `use-dns()` option is set to `no`), `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 `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, AxoSyslog 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 AxoSyslog server (`server`) as well, so if `relay` is configured properly, use the `keep-hostname(yes)` option on `server` to retain the proper HOST field. Setting `keep-hostname(no)` on `server` would result in AxoSyslog rewriting the HOST field to the address of the host that sent the message to `server`, which is `relay` in this case.
- If you cannot or do not want to resolve the `192.168.1.2` IP address on `relay`, but want to store your log messages on `server` using the IP address of the original host (that is, `client-host`), you can enable the `spoof-source()` option on `relay`. However, `spoof-source()` works only under the following conditions:
  - The `syslog-ng` binary has been compiled with the `--enable-spoof-source` option.
  - The log messages are sent using the highly unreliable UDP transport protocol. (Extremely unrecommended.)

3.4 Sending Kubernetes logs to OpenSearch

The following tutorial shows you how to install AxoSyslog on Kubernetes, deploy OpenSearch to the same cluster, and send Kubernetes logs to OpenSearch.

Prerequisites

You need:

- A Kubernetes cluster. We used minikube with docker driver and Helm. We used a Ubuntu 22.04 (amd64) machine, but it should work on any system that can run minikube (2 CPUs, 2GB of free memory, 20GB of free disk space).

  The OpenSearch service needs a large mmap count setting, so set it to at least 262144, for example:

  ```
  sysctl -w vm.max_map_count=262144
  ```

- Helm and kubectl installed.

Generate logs

If you don't already have an application that generates logs deployed to the Kubernetes cluster, install `kube-logging/log-generator` to generate sample logs. Complete the following steps.

1. Add the `kube-logging` repository to Helm.

   ```
   helm repo add kube-logging https://kube-logging.github.io/helm-charts
   ```

   Expected output:

   ```
   "kube-logging" has been added to your repositories
   ```

2. Update your repositories.

```
helm repo update
```

The output should look like:

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "kube-logging" chart repository
Update Complete. ⎈Happy Helming!⎈
```

3. Install `kube-logging/log-generator`.

```
helm install --generate-name --wait kube-logging/log-generator
```

The output should look like:

```
NAME: log-generator-1684694629
LAST DEPLOYED: Sun May 21 20:43:49 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

4. Check that the `log-generator` is running:

```
kubectl get pods
```

The output should look like:

```
NAME                                    READY   STATUS        RESTARTS
AGE
log-generator-1681984863-5946c559b9-ftrrn   1/1     Running       0
8s
```

Set up OpenSearch

1. Install an OpenSearch cluster with Helm:

```
helm repo add opensearch https://opensearch-project.github.io/helm-charts/
```

   Expected output:

```
"opensearch" has been added to your repositories
```

2. Update your repositories.

```
helm repo update
```

   The output should look like:

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "opensearch" chart repository
Update Complete. ⎈Happy Helming!⎈
```

3. Install OpenSearch. This step can take a few minutes.

```
helm install --generate-name --wait opensearch/opensearch
```

4. Install the OpenSearch dashboards.

```
helm install --generate-name --wait opensearch/opensearch-dashboards
```

   The output should look like:

```
NAME: opensearch-dashboards-1684695728
LAST DEPLOYED: Sun May 21 21:02:09 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l
"app.kubernetes.io/name=opensearch-
dashboards,app.kubernetes.io/instance=opensearch-dashboards-1684695728" -o
jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o
jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
```

5. Now you should have 5 pods. Check that they exist:

```
kubectl get pods
```

The output should look like:

```
NAME                                          READY   STATUS    RESTARTS
AGE
log-generator-1681984863-5946c559b9-ftrrn     1/1     Running   0
3m39s
opensearch-cluster-master-0                   1/1     Running   0
81s
opensearch-cluster-master-1                   1/1     Running   0
81s
opensearch-cluster-master-2                   1/1     Running   0
81s
opensearch-dashboards-1681999620-59f64f98f7-bjwwh   1/1     Running   0
44s
```

6. Forward the 5601 port of the OpenSearch Dashboards service (replace the name of the pod with your pod).

```
kubectl port-forward <name-of-your-opensearch-dashboards-pod> 8080:5601
```

   The output should look like:

```
Forwarding from 127.0.0.1:8080 -> 5601
Forwarding from [::1]:8080 -> 5601
```

7. Log in to the dashboard at `http://localhost:8080` with admin/admin. You will soon create an Index Pattern here, but first you have to send some logs from `syslog-ng`.

Set up axosyslog-collector

1. Add the AxoSyslog Helm repository:

```
helm repo add axosyslog https://axoflow.github.io/axosyslog
helm repo update
```

2. Create a YAML file (called `axoflow-demo.yaml` in the examples) to configure the collector.

```
config:
  sources:
    kubernetes:
      # Collect kubernetes logs
      enabled: true
  destinations:
    # Send logs to OpenSearch
    opensearch:
      - address: "opensearch-cluster-master"
        index: "test-axoflow-index"
        user: "admin"
        password: "admin"
        tls:
          # Do not validate the server's TLS certificate.
          peerVerify: false
        # Send the syslog fields + the metadata from .k8s.* in JSON format
        template: "$(format-json --scope rfc5424 --exclude DATE --key ISODATE
@timestamp=${ISODATE} k8s=$(format-json .k8s.* --shift-levels 2 --exclude
.k8s.log))"
```

3. Check how the `syslog-ng.conf` file looks with your custom values:

```
helm template -f axoflow-demo.yaml -s templates/config.yaml
axosyslog/axosyslog-collector
```

The output should look like:

```
# Source: axosyslog-collector/templates/config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    helm.sh/chart: axosyslog-collector-0.3.0
    app.kubernetes.io/name: axosyslog-collector
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "4.2.0"
    app.kubernetes.io/managed-by: Helm
  name: release-name-axosyslog-collector
data:
  syslog-ng.conf: |
    @version: current
    @include "scl.conf"

    options {
      stats(
        level(1)
      );
    };

    log {
      source { kubernetes(); };
      destination {
        elasticsearch-http(
          url("https://opensearch-cluster-master:9200/_bulk")
          index("test-axoflow-index")
          type("")
          template("$(format-json --scope rfc5424 --exclude DATE --key ISODATE
@timestamp=${ISODATE} k8s=$(format-json .k8s.* --shift-levels 2 --exclude
.k8s.log))")
          user("admin")
          password("admin")
          tls(
            peer-verify(no)
          )
        );
      };
    };
```

4. Install the `axosyslog-collector` chart:

```
helm install --generate-name --wait -f axoflow-demo.yaml axosyslog/axosyslog-collector
```

The output should look like:

```
NAME: axosyslog-collector-1682002179
LAST DEPLOYED: Thu Apr 20 16:49:39 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch the axosyslog-collector-1682002179 container start.
```

5. Check your pods:

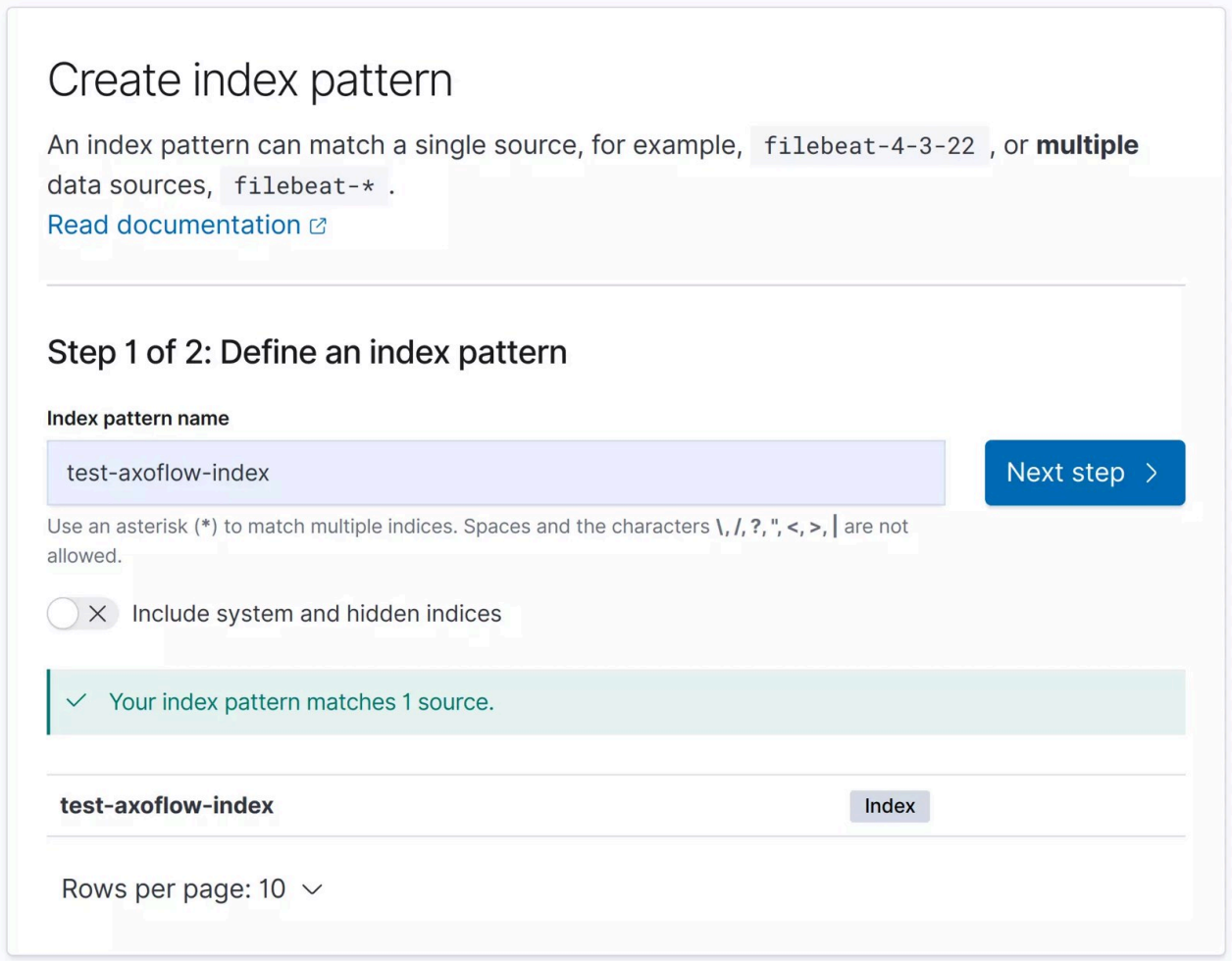
```
kubectl get pods --namespace=default -l app=axosyslog-collector-1682002179 -w
kubectl get pods
```

The output should look like:

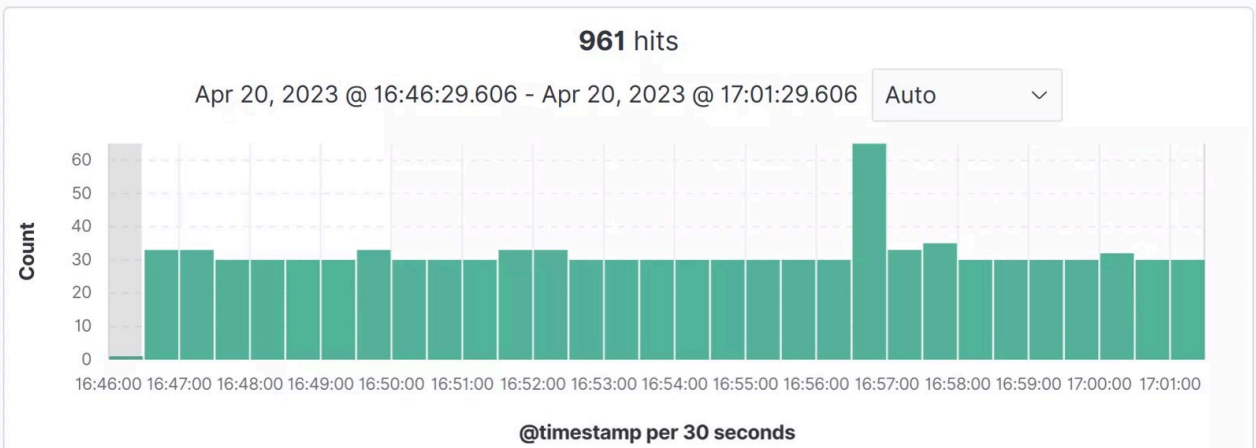
```
NAME                                              READY   STATUS    RESTARTS
AGE
log-generator-1681984863-5946c559b9-ftrrn         1/1     Running   0
13m
opensearch-cluster-master-0                       1/1     Running   0
11m
opensearch-cluster-master-1                       1/1     Running   0
11m
opensearch-cluster-master-2                       1/1     Running   0
11m
opensearch-dashboards-1681999620-59f64f98f7-bjwwh 1/1     Running   0
10m
axosyslog-collector-1682002179-pjlkn              1/1     Running   0
6s
```

Check the logs in OpenSearch

1. Open OpenSearch dashboard at `http://localhost:8080/app/management/opensearch-dashboards/`.
2. Create an Index Pattern called `test-axoflow-index`: `http://localhost:8080/app/management/opensearch-dashboards/indexPatterns`. At Step 2, set the **Time** field to `@timestamp`.



3. Now you can see your logs on the Discover view at `http://localhost:8080/app/discover`. Opening the detailed view for a log entry shows you the fields sent to OpenSearch.



Time	_source
Apr 20, 2023 @ 17:01:28.000	k8s.time: Apr 20, 2023 @ 17:01:28.254 k8s.stream: stdout k8s.pod_uuid: 4252773d-9abf-43c1-8cec-16d3316332b7 k8s.pod_name: log-generator-1681984863-5946c559b9-ftrrn k8s.namespace_name: default k8s.labels.pod-template-hash: 5946c559b9 k8s.labels.app.kubernetes.io/name: log-generator
Apr 20, 2023 @ 17:01:27.000	k8s.time: Apr 20, 2023 @ 17:01:27.254 k8s.stream: stdout k8s.pod_uuid: 4252773d-9abf-43c1-8cec-16d3316332b7 k8s.pod_name: log-generator-1681984863-5946c559b9-ftrrn k8s.namespace_name: default k8s.labels.pod-template-hash: 5946c559b9 k8s.labels.app.kubernetes.io/name: log-generator

@timestamp	Apr 20, 2023 @ 17:01:28.000
FACILITY	user
HOST	k8s/default/log-generator-1681984863-5946c559b9-ftrrn
ISODATE	Apr 20, 2023 @ 17:01:28.000
MESSAGE	38.156.24.196 - - [20/Apr/2023:15:01:28 +0000] "PUT /index.html HTTP/1.1" 404 5965 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2869.0 Safari/537.36" "-"
PID	4252773d-9abf-43c1-8cec-16d3316332b7/stdout
PRIORITY	notice
PROGRAM	log-generator
_id	Uwwvn4cB_3zkBCzJR2HM
_index	test-axoflow-index
_score	1
_type	-
k8s.container_hash	ghcr.io/kube-logging/log-generator@sha256:05aa441b20aaf42c14b57aeaf225fd69d0dc51d5e0e370dadc5356ef7535c13c
k8s.container_id	4036e82e1ae9
k8s.container_image	ghcr.io/kube-logging/log-generator:v0.4.1
k8s.container_name	log-generator
k8s.docker_id	4036e82e1ae98320c83aff15a51f848fe706444e83e16b39371723ddebf1d99f
k8s.labels.app.kubernetes.io/instance	log-generator-1681984863
k8s.labels.app.kubernetes.io/name	log-generator
k8s.labels.pod-template-hash	5946c559b9
k8s.namespace_name	default
k8s.pod_name	log-generator-1681984863-5946c559b9-ftrrn
k8s.pod_uuid	4252773d-9abf-43c1-8cec-16d3316332b7
k8s.stream	stdout
k8s.time	Apr 20, 2023 @ 17:01:28.254

3.5 Managing and checking the syslog-ng service on Linux

Start, stop, reload, and check the status of the AxoSyslog service on Linux.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

This section describes how to start, stop and check the status of AxoSyslog service on Linux.

Starting AxoSyslog

To start AxoSyslog, execute the following command as root. For example:

```
systemctl start syslog-ng
```

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

The following message indicates that AxoSyslog can not start (see Checking AxoSyslog status):

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

Stop AxoSyslog

To stop AxoSyslog

1. Execute the following command as root.

   ```
   systemctl stop syslog-ng
   ```

2. Check the status of AxoSyslog service (see Checking AxoSyslog status).

Restart AxoSyslog

To restart AxoSyslog, execute the following command as root.

```
systemctl restart syslog-ng
```

Reload configuration file without restarting AxoSyslog

To reload the configuration file without restarting AxoSyslog, execute the following command as root.

```
systemctl reload syslog-ng
```

Check AxoSyslog status

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

Check the status of AxoSyslog service

To check the status of AxoSyslog service

1. Execute the following command as root.

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

2. Check the `Active:` field, which shows the status of AxoSyslog service. The following statuses are possible:

   - `active (running)` - `syslog-ng` service is up and running

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

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

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

Check the process of AxoSyslog

To check the process of AxoSyslog, execute one of the following commands.

- `ps u <pid of syslog-ng>`

  Expected output example:

```
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
syslogng 6709 0.0 0.6 308680 13432 ? Ss 09:17 0:00 /opt/syslog-
ng/libexec/syslog-ng -F --no-caps --enable-core
```

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

  Expected output example:

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

Check the internal logs of AxoSyslog

The internal logs of AxoSyslog contains informal, warning and error messages.

By default, AxoSyslog log messages (generated on the `internal()` source) are written to `/var/log/messages`.

Check the internal logs of AxoSyslog for any issue.

> **Note**
>
> If you find error messages in the internal logs, messages aren't processed by AxoSyslog, or you encounter any issue, you have the following options:
>
> - Open a GitHub issue including the results.
> - contact us

Monitor AxoSyslog

To monitor the performance and status of AxoSyslog for observability and monitoring, see Statistics and metrics of AxoSyslog.

4 Install AxoSyslog

Install AxoSyslog on Debian, Ubuntu, RHEL, Fedora, or AlmaLinux, run it with Docker, Podman, or Helm, upgrade to a newer version, or upgrade from syslog-ng.

This chapter explains how to install AxoSyslog on various platforms.

Cloud-ready `syslog-ng` images

AxoSyslog provides cloud-ready images . These images differ from the upstream `syslog-ng` images, because:

- They're based on Alpine Linux, instead of Debian testing for reliability and smaller size (thus smaller attack surface).
- They incorporate cloud-native features and settings, such as the Kubernetes source.
- They incorporate container-level optimizations for better performance and improved security. For example, they use an alternative malloc library.
- They support the ARM architecture.

The AxoSyslog images support the following architectures:

- amd64
- arm/v7
- arm64

Install AxoSyslog

- Install AxoSyslog with Podman and systemd
- Install AxoSyslog on Debian/Ubuntu
- Install AxoSyslog on RHEL/Fedora/AlmaLinux
- Install AxoSyslog with Docker
- Install AxoSyslog with Podman
- Install AxoSyslog with Helm
- AxoSyslog on macOS

Upgrade AxoSyslog

To upgrade an existing AxoSyslog installation to a newer version (packages, container images, or the Helm chart), see Upgrade AxoSyslog to a newer version.

Upgrade `syslog-ng` to AxoSyslog

If you're already using `syslog-ng`, you can upgrade your existing `syslog-ng` deployments to AxoSyslog in a matter of minutes. For details, see Upgrade syslog-ng to AxoSyslog.

Forward Windows logs

Axoflow provides a custom OpenTelemetry Collector distribution that you can use to collect logs on Windows hosts and forward them to AxoSyslog. For details, see Forward Windows logs.

4.1 Install AxoSyslog with Podman and systemd

This page shows you how to run AxoSyslog as a systemd service using podman.

AxoSyslog provides cloud-ready images . These images differ from the upstream `syslog-ng` images, because:

- They're based on Alpine Linux, instead of Debian testing for reliability and smaller size (thus smaller attack surface).
- They incorporate cloud-native features and settings, such as the Kubernetes source.
- They incorporate container-level optimizations for better performance and improved security. For example, they use an alternative malloc library.
- They support the ARM architecture.

The AxoSyslog images support the following architectures:

- amd64
- arm/v7
- arm64

Prerequisites

Podman version 4.6.1.

The steps in this procedure were tested on CentOS 9, but should work on other similar distributions as well.

Install AxoSyslog as a systemd service

1. Make sure that there is no `axosyslog.service` unit file on the system. Run the following commands:

   ```
   sudo rm /etc/systemd/system/axosyslog.service
   ```

   Expected output:

   ```
   rm: cannot remove '/etc/systemd/system/axosyslog.service': No such file or
   directory
   ```

```
sudo systemctl cat axosyslog.service
```

Expected output:

```
No files found for axosyslog.service.
```

2. Create a systemd unit file called `/etc/containers/systemd/axosyslog.container` based on the following template:

```
sudo curl -o /etc/containers/systemd/axosyslog.container
https://axoflow.com/docs/axosyslog-core/install/podman-
systemd/axosyslog.container
```

```

```

3. Edit the unit file as needed for your environment.

   We recommend using the default mount points:

Purpose	On the host	In the container
Disk-buffer and persist files	`/var/lib/syslog-ng`	`/var/lib/syslog-ng`
`syslog-ng` configuration file	`/opt/axosyslog/etc`	`/etc/syslog-ng`
Output log files	`/opt/axosyslog/var/log`	`/var/log`

4. (Optional) Create an `override.conf` file to set custom environment values. This can be useful if you don't want to modify `/etc/containers/systemd/axosyslog.container`. Run:

```
systemctl edit axosyslog
```

   Later you can edit this file by running the previous command again.

5. Create the `/opt/axosyslog/etc/syslog-ng.conf` configuration file based on the following template.

```
sudo mkdir -p /opt/axosyslog/etc/ ; sudo curl -o /opt/axosyslog/etc/syslog-ng.conf https://axoflow.com/docs/axosyslog-core/install/podman-systemd/syslog-ng.conf
```

With the following sample configuration file AxoSyslog collects the local system logs and logs received from the network into the `/var/log/messages` file.

```

```

You can customize the configuration file according to your needs. For a few pointers, see Configuring AxoSyslog as a logserver and the rest of this guide.

6. Run the following commands to reload the systemd configuration and launch the `axosyslog` service. Though the systemctl commands are run as root, the container will run as the specified user if set appropriately in the unit file.

```
sudo systemctl daemon-reload
sudo systemctl stop axosyslog
sudo systemctl start axosyslog
```

If there aren't any errors, these commands don't have any output.

7. Run the following command to verify that the service was properly started:

```
journalctl -b -u axosyslog | tail -100
```

The output should be similar to:

```
Feb 12 09:04:40 <your-hostname> systemd[1]: Starting AxoSyslog Container...
Feb 12 09:04:40 <your-hostname> podman[2783]: 2024-02-12 09:04:40.454665314 -0500 EST m=+0.167732500 system refresh
Feb 12 09:04:40 <your-hostname> axosyslog[2783]: Trying to pull ghcr.io/axoflow/axosyslog:latest...
Feb 12 09:04:40 <your-hostname> axosyslog[2783]: Pulling image //ghcr.io/axoflow/axosyslog:latest inside systemd: setting pull timeout to 5m0s
```

```
Feb 12 09:04:41 <your-hostname> axosyslog[2783]: Getting image source
signatures
Feb 12 09:04:41 <your-hostname> axosyslog[2783]: Copying blob
sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1
Feb 12 09:04:41 <your-hostname> axosyslog[2783]: Copying blob
sha256:619be1103602d98e1963557998c954c892b3872986c27365e9f651f5bc27cab8
Feb 12 09:04:41 <your-hostname> axosyslog[2783]: Copying blob
sha256:b061f41886afb563aff2a5f731f3286ba54ea6f657ed3e282f5339a12a64c5ef
Feb 12 09:04:41 <your-hostname> axosyslog[2783]: Copying blob
sha256:1b8d965a650c6a05227bd5c549930c9898071e8e7abb26886d4169a99762de0a
Feb 12 09:04:41 <your-hostname> axosyslog[2783]: Copying blob
sha256:b5b0ce6ebef193c4f909379188cfb59443e8a1809816fbb476074908b170b4d1
Feb 12 09:04:50 <your-hostname> axosyslog[2783]: Copying config
sha256:c379d94ef2c5ec348dfb3a93eed9a19aed667c396008db85edc354c8f4f8cb6a
Feb 12 09:04:50 <your-hostname> axosyslog[2783]: Writing manifest to image
destination
Feb 12 09:04:50 <your-hostname> podman[2783]: 2024-02-12 09:04:50.422390687
-0500 EST m=+10.135457863 container create
477c9f011684f767aae138a0f88602ff30a8c95a46d616bb3b95318ec3a4b79f
(image=ghcr.io/axoflow/axosyslog:latest, name=AxoSyslog,
org.opencontainers.image.documentation=https://axoflow.com/docs/axosyslog/docs/
, org.opencontainers.image.url=https://axoflow.io/,
org.opencontainers.image.source=https://github.com/axoflow/axosyslog,
org.opencontainers.image.authors=Axoflow,
org.opencontainers.image.title=AxoSyslog,
org.opencontainers.image.vendor=Axoflow, PODMAN_SYSTEMD_UNIT=axosyslog.service,
org.opencontainers.image.description=A cloud-native distribution of syslog-ng
by Axoflow, maintainer=axoflow.io, org.opencontainers.image.licenses=GPL-3.0-
only)
Feb 12 09:04:50 <your-hostname> podman[2783]: 2024-02-12 09:04:50.402626446
-0500 EST m=+10.115693622 image pull
c379d94ef2c5ec348dfb3a93eed9a19aed667c396008db85edc354c8f4f8cb6a
ghcr.io/axoflow/axosyslog:latest
Feb 12 09:04:50 <your-hostname> podman[2783]: 2024-02-12 09:04:50.489925509
-0500 EST m=+10.202992695 container init
477c9f011684f767aae138a0f88602ff30a8c95a46d616bb3b95318ec3a4b79f
(image=ghcr.io/axoflow/axosyslog:latest, name=AxoSyslog,
org.opencontainers.image.authors=Axoflow,
org.opencontainers.image.licenses=GPL-3.0-only,
org.opencontainers.image.vendor=Axoflow, maintainer=axoflow.io,
PODMAN_SYSTEMD_UNIT=axosyslog.service,
org.opencontainers.image.url=https://axoflow.io/,
org.opencontainers.image.documentation=https://axoflow.com/docs/axosyslog/docs/
```

```
, org.opencontainers.image.title=AxoSyslog,
org.opencontainers.image.description=A cloud-native distribution of syslog-ng
by Axoflow,
org.opencontainers.image.source=https://github.com/axoflow/axosyslog)Feb 12
09:04:50 <your-hostname> systemd[1]: Started AxoSyslog Container.
Feb 12 09:04:50 <your-hostname> podman[2783]: 2024-02-12 09:04:50.500050669
-0500 EST m=+10.213117845 container start
477c9f011684f767aae138a0f88602ff30a8c95a46d616bb3b95318ec3a4b79f
(image=ghcr.io/axoflow/axosyslog:latest, name=AxoSyslog,
PODMAN_SYSTEMD_UNIT=axosyslog.service,
org.opencontainers.image.source=https://github.com/axoflow/axosyslog,
org.opencontainers.image.authors=Axoflow,
org.opencontainers.image.description=A cloud-native distribution of syslog-ng
by Axoflow,
org.opencontainers.image.documentation=https://axoflow.com/docs/axosyslog/docs/
, org.opencontainers.image.licenses=GPL-3.0-only,
org.opencontainers.image.vendor=Axoflow,
org.opencontainers.image.title=AxoSyslog, maintainer=axoflow.io,
org.opencontainers.image.url=https://axoflow.io/)
Feb 12 09:04:50 <your-hostname> axosyslog[2783]:
477c9f011684f767aae138a0f88602ff30a8c95a46d616bb3b95318ec3a4b79f
Feb 12 09:04:50 <your-hostname> AxoSyslog[2821]: [2024-02-12T14:04:50.806054]
syslog-ng starting up; version='4.6.0'
```

8. Send a test message to the service (requires `nc` to be installed):

```
echo '<5> localhost test: this is a test message' | nc localhost 514
```

Check that the test message has arrived into the log file:

```
cat /opt/axosyslog/var/log/messages
```

The output should be similar to:

```
Feb 19 15:49:12 localhost test: this is a test message
```

Customize the configuration

To customize the configuration, edit the `/opt/axosyslog/etc/syslog-ng.conf` file on the host, then reload the service.

Managing the AxoSyslog systemd service

- You can reload `syslog-ng` running in the container via systemctl. The following command reloads the `syslog-ng.conf` file, without stopping/starting `syslog-ng` itself.

  ```
  sudo systemctl reload axosyslog
  ```

- You can access `syslog-ng-ctl` from the host, for example by running:

  ```
  podman exec -ti AxoSyslog syslog-ng-ctl show-license-info
  ```

  If you use `syslog-ng-ctl` regularly, you can create the `/opt/axosyslog/bin/syslog-ng-ctl` file with the following content, make it executable, and add it to your path. That way running `syslog-ng-ctl <command>` will execute the command in the AxoSyslog container.

  ```
  #!/bin/bash

  podman exec -ti AxoSyslog syslog-ng-ctl "$@"
  ```

- The traditional method of starting a service at boot (`systemctl enable`) is not supported for container services. To automatically start the AxoSyslog service, make sure that the following line is included in the unit file. (It is included in the sample template.)

  ```
  [Install]
  WantedBy=default.target
  ```

4.2 Install AxoSyslog on Debian/Ubuntu

You can install AxoSyslog 4.8 and newer on your Debian-based system from Axoflow's APT repository. AxoSyslog is a drop in replacement for the `syslog-ng` Debian package , all the AxoSyslog binaries and configuration files are stored at the same place on your system.

The following x86-64 distributions are supported:

Distribution	sources.list component
Debian 13 (x86-64)	debian-trixie
Debian 12 (x86-64)	debian-bookworm
Debian 11 (x86-64)	debian-bullseye
Debian Unstable (x86-64)	debian-sid
Debian Testing (x86-64)	debian-testing
Ubuntu 26.04 (x86-64)	ubuntu-resolute
Ubuntu 25.04 (x86-64)	ubuntu-plucky
Ubuntu 24.04 (x86-64)	ubuntu-noble
Ubuntu 22.04 (x86-64)	ubuntu-jammy
Ubuntu 20.04 (x86-64)	ubuntu-focal

Which package to install?

AxoSyslog supports many features, but you rarely need all of them on a single host. Sources and destinations that depend on external libraries live in separate modules, so you install only the ones you actually use. For example, the gRPC-based destinations (like loki() and opentelemetry()) come from the gRPC module, while HTTP-based destinations (like elasticsearch-http() and sumologic-http()) come from the HTTP module.

The Prerequisites section of every source, destination, and parser names the module it needs.

If a module isn't installed, AxoSyslog doesn't start, and reports a syntax error that points at the name of the driver you configured. For details, see Error: unexpected LL_IDENTIFIER .

The following table lists the AxoSyslog modules, the configuration objects each provides, and the package to install.

Module	Provides	Package
Base	`file()`, `network()`, `syslog()`, `tcp()`, `udp()`, `unix-stream()`, `unix-dgram()`, `pipe()`, `program()`, `stdin()`, `stdout()`, `usertty()`, `wildcard-file()`, `pseudofile()`, `system()`, `systemd-journal()`, `systemd-syslog()`, `internal()`, `csv-parser()`, `db-parser()`, `json-parser()`, `kv-parser()`, `linux-audit-parser()`, `date-parser()`, `regexp-parser()`, `tags-parser()`, `syslog-parser()`, `sdata-parser()`, `group-lines()`, `grouping-by()`, `app-parser()`, `metrics-probe()`, `disk-buffer()`, `rate-limit()`, and most template functions	`axosyslog-core`
Configuration Library (SCL)	`linux-audit()`, `default-network-drivers()`, `mbox()`, `nodejs()`, `osquery()`, `pacct()`, `snmptrap()`, `jellyfin()`, `pihole-ftl()`, `qbittorrent()`, `radarr()` and the other `*arr()` sources, `collectd()`, `graylog2()`, `loggly()`, `logmatic()`, `syslog-ng()`, `ewmm()`, the application adapters (`apache-accesslog-parser()`, `cisco-parser()`, `panos-parser()`, `sudo-parser()`, and so on), and every HTTP-based destination	`axosyslog-scl`
gRPC	`opentelemetry()`, `axosyslog-otlp()` (formerly `syslog-ng-otlp()`), `loki()`, `bigquery()`, `clickhouse()`, `google-pubsub-grpc()`, the `otel_*()` and `protobuf_message()` FilterX functions	`axosyslog-mod-grpc`
HTTP	`http()` destination, `ehttp()`, `elasticsearch-bulk()`, and `splunk-hec()` sources, `azure-auth-header()`, and all SCL destinations built on HTTP: `elasticsearch-http()`, `elasticsearch-datastream()`, `opensearch()`, `openobserve-log()`, `logscale()`, `splunk-hec-event()`, `splunk-hec-raw()`, `sumologic-http()`, `slack()`, `discord()`, `telegram()`, `azure-monitor()`, `google-pubsub()`	`axosyslog-mod-http`
Python	`python()` source, destination, parser and template function, `python-fetcher()`, `python-http-header()`, and the Python-based SCL drivers `kubernetes()`, `kubernetes-metadata-parser()`, `s3()`, `webhook()`, `webhook-json()`, `hypr-app-audit-trail()`	`axosyslog-mod-python`
Cloud authentication	`cloud-auth()`, used by `azure-monitor()` and `google-pubsub()`	`axosyslog-mod-cloud-auth`

Kafka	`kafka-c()` and the `kafka()` SCL destination	`axosyslog-mod-rdkafka`
MQTT	`mqtt()` source and destination	`axosyslog-mod-mqtt`
AMQP	`amqp()`	`axosyslog-mod-amqp`
MongoDB	`mongodb()`	`axosyslog-mod-mongodb`
SQL	`sql()`	`axosyslog-mod-sql`
Redis	`redis()`	`axosyslog-mod-redis`
Riemann	`riemann()`	`axosyslog-mod-riemann`
SMTP	`smtp()`	`axosyslog-mod-smtp`
SNMP	`snmp()` , `snmptrapd-parser()` , and the `snmptrap()` SCL source	`axosyslog-mod-snmp`
GeoIP2	`geoip2()` parser and the `$(geoip2)` template function	`axosyslog-mod-geoip2`
Java	`java()` . Only the HDFS Java module is shipped, the Java implementations of the Elasticsearch and HTTP destinations aren't.	`axosyslog-mod-java` , `axosyslog-mod-java-common-lib`
HDFS	`hdfs()`	`axosyslog-mod-hdfs` (also requires `axosyslog-mod-java` and `axosyslog-mod-java-common-lib`)
Secure logging	`$(slog)` template function and the `slog*` command-line tools	`axosyslog-mod-slog`
eBPF	`ebpf()`	`axosyslog-mod-bpf`
XML parser	`xml()` , `windows-eventlog-xml-parser()` , and the `parse_xml()` , `format_xml()` , `parse_windows_eventlog_xml()` , `format_windows_eventlog_xml()` FilterX functions	`axosyslog-mod-xml-parser`
STOMP	`stomp()`	`axosyslog-mod-stomp`

Graphite	`$(graphite-output)` template function and the `graphite()` SCL destination	`axosyslog-mod-graphite`
add-contextual-data	`add-contextual-data()`	`axosyslog-mod-add-contextual-data`
map-value-pairs	`map-value-pairs()`	`axosyslog-mod-map-value-pairs`
getent	`$(getent)` template function	`axosyslog-mod-getent`
stardate	`$(stardate)` template function	`axosyslog-mod-stardate`
Examples	`example-msg-generator()`, `example-random-generator()`, `random-choice-generator()`, `example-destination()`	`axosyslog-mod-examples`
Apache Arrow Flight	`arrow-flight()`	`axosyslog-mod-arrow-flight`

The `axosyslog` metapackage installs `axosyslog-core`, `axosyslog-scl`, and recommends every optional module, so `apt install axosyslog` gives you a working setup with the common modules.

AxoSyslog supports the `sun-streams()`, `darwin-oslog()`, `darwin-oslog-stream()`, and `openbsd()` drivers only on Solaris, macOS, and OpenBSD respectively. The Debian/Ubuntu and RHEL packages don't include them.

Usually, you install the base package `axosyslog`, and the packages of specific modules that you want to use. We also provide `debuginfo` packages for every module, but you only need these in certain troubleshooting scenarios.

Steps

To install AxoSyslog from the APT repository, complete the following steps.

1. Run the following commands to add the APT repository of your distribution (for example, Ubuntu 24.04) to the APT sources list:

```
wget -qO - https://pkg.axoflow.io/axoflow-code-signing-pub.asc | gpg --dearmor
> /usr/share/keyrings/axoflow-code-signing-pub.gpg
```

```
echo "deb [signed-by=/usr/share/keyrings/axoflow-code-signing-pub.gpg]
https://pkg.axoflow.io/apt stable ubuntu-noble" | tee --append
/etc/apt/sources.list.d/axoflow.list
```

```
apt update
```

> **Note**
>
> Nightly builds are also available:
>
> ```
> echo "deb [signed-by=/usr/share/keyrings/axoflow-code-signing-pub.gpg]
> https://pkg.axoflow.io/apt nightly ubuntu-noble" | tee --append
> /etc/apt/sources.list.d/axoflow.list
> ```

2. Install the AxoSyslog package.

```
apt install axosyslog
```

Using AxoSyslog

After you've installed AxoSyslog, you can configure it just like `syslog-ng`, using the same configurations files (`/etc/syslog-ng/syslog-ng.conf` by default). For details, see the Quick-start guide.

Getting help

If you run into any issues while installing or configuring AxoSyslog, or you have any questions, you can find us on our Discord server .

4.3 Install AxoSyslog on RHEL/Fedora/AlmaLinux

You can install AxoSyslog 4.8 and newer on your RPM-based system from Axoflow's RPM repository. AxoSyslog is a drop in replacement for the `syslog-ng` RPM package, all the AxoSyslog binaries and configuration files are stored at the same place on your system.

The following distributions are supported:

- Red Hat Enterprise Linux (RHEL) 10 x86-64 / AlmaLinux 10 x86-64
- Red Hat Enterprise Linux (RHEL) 9 x86-64 / AlmaLinux 9 x86-64
- Red Hat Enterprise Linux (RHEL) 8 x86-64 / AlmaLinux 8 x86-64
- Fedora 44 x86-64

(The packages for AlmaLinux probably work for Rocky Linux as well, but we haven't tested it.)

Which package to install?

AxoSyslog supports many features, but you rarely need all of them on a single host. Sources and destinations that depend on external libraries live in separate modules, so you install only the ones you actually use. For example, the gRPC-based destinations (like loki() and opentelemetry()) come from the gRPC module, while HTTP-based destinations (like elasticsearch-http() and sumologic-http()) come from the HTTP module.

The Prerequisites section of every source, destination, and parser names the module it needs.

If a module isn't installed, AxoSyslog doesn't start, and reports a syntax error that points at the name of the driver you configured. For details, see Error: unexpected LL_IDENTIFIER .

The following table lists the AxoSyslog modules, the configuration objects each provides, and the package to install.

Module	Provides	Package

Base	`file()`, `network()`, `syslog()`, `tcp()`, `udp()`, `unix-stream()`, `unix-dgram()`, `pipe()`, `program()`, `stdin()`, `stdout()`, `usertty()`, `wildcard-file()`, `pseudofile()`, `system()`, `systemd-journal()`, `systemd-syslog()`, `internal()`, `csv-parser()`, `db-parser()`, `json-parser()`, `kv-parser()`, `linux-audit-parser()`, `date-parser()`, `regexp-parser()`, `tags-parser()`, `syslog-parser()`, `sdata-parser()`, `group-lines()`, `grouping-by()`, `app-parser()`, `metrics-probe()`, `disk-buffer()`, `rate-limit()`, and most template functions	`axosyslog`
Configuration Library (SCL)	`linux-audit()`, `default-network-drivers()`, `mbox()`, `nodejs()`, `osquery()`, `pacct()`, `snmptrap()`, `jellyfin()`, `pihole-ftl()`, `qbittorrent()`, `radarr()` and the other `*arr()` sources, `collectd()`, `graylog2()`, `loggly()`, `logmatic()`, `syslog-ng()`, `ewmm()`, the application adapters (`apache-accesslog-parser()`, `cisco-parser()`, `panos-parser()`, `sudo-parser()`, and so on), and every HTTP-based destination	Part of the `axosyslog` base package
gRPC	`opentelemetry()`, `axosyslog-otlp()` (formerly `syslog-ng-otlp()`), `loki()`, `bigquery()`, `clickhouse()`, `google-pubsub-grpc()`, the `otel_*()` and `protobuf_message()` FilterX functions	`axosyslog-grpc`
HTTP	`http()` destination, `ehttp()`, `elasticsearch-bulk()`, and `splunk-hec()` sources, `azure-auth-header()`, and all SCL destinations built on HTTP: `elasticsearch-http()`, `elasticsearch-datastream()`, `opensearch()`, `openobserve-log()`, `logscale()`, `splunk-hec-event()`, `splunk-hec-raw()`, `sumologic-http()`, `slack()`, `discord()`, `telegram()`, `azure-monitor()`, `google-pubsub()`	`axosyslog-http`
Python	`python()` source, destination, parser and template function, `python-fetcher()`, `python-http-header()`, and the Python-based SCL drivers `kubernetes()`, `kubernetes-metadata-parser()`, `s3()`, `webhook()`, `webhook-json()`, `hypr-app-audit-trail()`	`axosyslog-python`
Cloud authentication	`cloud-auth()`, used by `azure-monitor()` and `google-pubsub()`	`axosyslog-cloud-auth`
Kafka	`kafka-c()` and the `kafka()` SCL destination	`axosyslog-kafka`
MQTT	`mqtt()` source and destination	`axosyslog-mqtt`

AMQP	`amqp()`	`axosyslog-amqp`
MongoDB	`mongodb()`	`axosyslog-mongodb`
SQL	`sql()`	`axosyslog-sql`
Redis	`redis()`	`axosyslog-redis`
Riemann	`riemann()`	`axosyslog-riemann`
SMTP	`smtp()`	`axosyslog-smtp`
SNMP	`snmp()`, `snmptrapd-parser()`, and the `snmptrap()` SCL source	`axosyslog-afsnmp`
GeoIP2	`geoip2()` parser and the `$(geoip2)` template function	`axosyslog-geoip`
Java	`java()`. Only the HDFS Java module is shipped, the Java implementations of the Elasticsearch and HTTP destinations aren't.	`axosyslog-java`
HDFS	`hdfs()`	`axosyslog-java`
Secure logging	`$(slog)` template function and the `slog*` command-line tools	`axosyslog-slog`
eBPF	`ebpf()`	`axosyslog-bpf`
XML parser	`xml()`, `windows-eventlog-xml-parser()`, and the `parse_xml()`, `format_xml()`, `parse_windows_eventlog_xml()`, `format_windows_eventlog_xml()` FilterX functions	Part of the `axosyslog` base package
STOMP	`stomp()`	Part of the `axosyslog` base package
Graphite	`$(graphite-output)` template function and the `graphite()` SCL destination	Part of the `axosyslog` base package
add-contextual-data	`add-contextual-data()`	Part of the `axosyslog` base package

map-value-pairs	`map-value-pairs()`	Part of the `axosyslog` base package
getent	`$(getent)` template function	Part of the `axosyslog` base package
stardate	`$(stardate)` template function	Part of the `axosyslog` base package
Examples	`example-msg-generator()`, `example-random-generator()`, `random-choice-generator()`, `example-destination()`	Part of the `axosyslog` base package
Apache Arrow Flight	`arrow-flight()`	Not available

Note that the RPM package names differ from the Debian package names: they don't have the `mod-` part, and some of them use a different name (for example, the GeoIP2 module is `axosyslog-geoip`, and the Kafka module is `axosyslog-kafka`).

AxoSyslog supports the `sun-streams()`, `darwin-oslog()`, `darwin-oslog-stream()`, and `openbsd()` drivers only on Solaris, macOS, and OpenBSD respectively. The Debian/Ubuntu and RHEL packages don't include them.

Usually, you install the base package `axosyslog-<version-number>.<distro>.x86_64.rpm`, and the packages of specific modules that you want to use. We also provide `debuginfo` packages for every module, but you only need these in certain troubleshooting scenarios.

Steps

To install AxoSyslog on RedHat Enterprise Linux 9 or AlmaLinux 9, complete the following steps. The instructions for AlmaLinux probably work for Rocky Linux 9 as well, but we haven't tested it.

1. Run the following commands to enable the EPEL repositories for your distribution. This is needed to install some dependencies of AxoSyslog. (For RHEL 8 and compatible distributions, use these instructions .)

    - RHEL 9-10:

```
sudo subscription-manager repos --enable codeready-builder-for-rhel-
9-$(arch)-rpms
sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-
9.noarch.rpm
```

- AlmaLinux 9-10:

```
sudo dnf install epel-release
sudo dnf config-manager --set-enabled crb
```

- Fedora:

```
sudo dnf install epel-release
```

2. Add the AxoSyslog repository of your distribution:

RHEL / AlmaLinux 9

```
sudo tee /etc/yum.repos.d/axosyslog.repo <<< '[axosyslog]
name=AxoSyslog
baseurl=https://pkg.axoflow.io/rpm/stable/almalinux-9/$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://pkg.axoflow.io/axoflow-code-signing-pub.asc' > /dev/null
```

RHEL / AlmaLinux 8

```
sudo tee /etc/yum.repos.d/axosyslog.repo <<< '[axosyslog]
name=AxoSyslog
baseurl=https://pkg.axoflow.io/rpm/stable/almalinux-8/$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://pkg.axoflow.io/axoflow-code-signing-pub.asc' > /dev/null
```

Fedora 41

```
sudo tee /etc/yum.repos.d/axosyslog.repo <<< '[axosyslog]
name=AxoSyslog
baseurl=https://pkg.axoflow.io/rpm/stable/fedora-41/$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://pkg.axoflow.io/axoflow-code-signing-pub.asc' > /dev/null
```

Fedora 40

```
sudo tee /etc/yum.repos.d/axosyslog.repo <<< '[axosyslog]
name=AxoSyslog
baseurl=https://pkg.axoflow.io/rpm/stable/fedora-40/$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://pkg.axoflow.io/axoflow-code-signing-pub.asc' > /dev/null
```

Fedora 39

```
sudo tee /etc/yum.repos.d/axosyslog.repo <<< '[axosyslog]
name=AxoSyslog
baseurl=https://pkg.axoflow.io/rpm/stable/fedora-39/$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://pkg.axoflow.io/axoflow-code-signing-pub.asc' > /dev/null
```

3. Update the packages list.

```
sudo yum update -y
```

Expected output:

```
AxoSyslog
544  B/s | 488  B     00:00
AxoSyslog
5.2 kB/s | 3.2 kB     00:00
Importing GPG key 0x5F25E107:
Userid     : "Axoflow Code Signing Key <support@axoflow.com>"
Fingerprint: 365A 4340 FA76 89B4 78ED 617C 3605 FFAD 5F25 E107
From       : https://pkg.axoflow.io/axoflow-code-signing-pub.asc
AxoSyslog
68 kB/s |  56 kB     00:00
Extra Packages for Enterprise Linux 9 - x86_64
8.2 MB/s |  23 MB     00:02
Extra Packages for Enterprise Linux 9 openh264 (From Cisco) - x86_64
1.1 kB/s | 2.5 kB     00:02
Dependencies resolved.
Nothing to do.
Complete!
```

4. Install AxoSyslog.

   - To install AxoSyslog with every available module, run:

```
sudo yum install axosyslog-*
```

- To install only the base package, run:

```
sudo yum install axosyslog
```

Then install other packages for the modules you want to use as needed. For example, to use the gRPC-based destinations (like loki() or opentelemetry()), install the `axosyslog-grpc-*` package. For HTTP-based destinations like elasticsearch-http() or sumologic-http() , you need the `axosyslog-http-*` package.

5. Enable `syslog-ng` .

```
sudo systemctl enable syslog-ng
sudo systemctl start syslog-ng
```

6. (Optional) If you don't want to run other log collectors on the host, you can delete the existing one (which is rsyslog by default):

```
sudo yum remove rsyslog.x86_64
```

Using AxoSyslog

After you've installed AxoSyslog, you can configure it just like `syslog-ng` , using the same configurations files (`/etc/syslog-ng/syslog-ng.conf` by default). For details, see the Quick-start guide.

Getting help

If you run into any issues while installing or configuring AxoSyslog, or you have any questions, you can find us on our Discord server .

4.4 Install AxoSyslog with Helm

Install AxoSyslog on Kubernetes with the Helm chart, as a collector, an aggregator, or both.

AxoSyslog provides a Helm chart . You can use this chart to install cloud-ready `syslog-ng` images created and maintained by Axoflow .

Prerequisites for the Helm chart

To use this chart, you need:

- A Kubernetes cluster that runs Kubernetes 1.22 or newer.
- `kubectl` , configured to access your cluster.
- Helm 3.0 or newer . For details, see the official Helm documentation .

Collector and aggregator use cases

The chart has parameters for the following use cases:

- As a collector : The collector uses the `kubernetes()` source to collect the local logs. It sends the logs to the aggregator, to a syslog server, to an OpenSearch node, or to a different AxoSyslog node.
- As an aggregator : The aggregator receives RFC3164 and RFC5424 syslog messages from all senders, and `axosyslog-otlp()` messages from other AxoSyslog nodes. It stores the messages locally, or sends them to remote destinations.

By default, the collector sends the logs to the aggregator. You can configure and deploy the two components independently. To use other sources and destinations, use the `config.raw` parameter of the collector or the aggregator. For the list of parameters and their default values, see Parameters of the AxoSyslog Helm chart.

Install the Helm chart

To install the `axosyslog` chart, complete the following steps.

1. Add the chart repository.

```
helm repo add axosyslog https://axoflow.github.io/axosyslog
helm repo update
```

2. Install the chart. The default settings install the following into the `default` namespace:

   - A `collector` DaemonSet. It collects the pod logs on each node, and sends them to the aggregator.
   - An `aggregator` StatefulSet. It receives syslog and `axosyslog-otlp()` messages, and writes them to a file.

   To install only one component, disable the other component: set `collector.enabled=false` or `aggregator.enabled=false`. For the list of parameters and their default values, see Parameters of the AxoSyslog Helm chart. If you use disk-buffers, also read How to use disk-buffers in containers and Kubernetes .

   - Install with the default values:

     ```
     helm install --generate-name axosyslog/axosyslog
     ```

   - Install only the collector:

     ```
     helm install --generate-name axosyslog/axosyslog --set
     aggregator.enabled=false
     ```

   - Install only the aggregator:

     ```
     helm install --generate-name axosyslog/axosyslog --set
     collector.enabled=false
     ```

   The output should be similar to:

```
NAME: axosyslog-1713953907
LAST DEPLOYED: Wed Apr 24 12:18:28 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Watch the axosyslog-1713953907 containers start.
  $ kubectl get pods --namespace=default -l
app.kubernetes.io/instance=axosyslog-
1713953907,app.kubernetes.io/name=axosyslog -w
```

The `NAME` field shows the name of the release. In the following commands, replace `<release-name>` with this name.

3. Check that the pods are running.

```
kubectl get pods
```

The output should list the pods that are running: a collector pod on every node and an aggregator pod for the default settings. For example, on a single-node cluster:

```
NAME                                  READY   STATUS    RESTARTS   AGE
axosyslog-1713953907-collector-ddftq   1/1     Running   0          57s
axosyslog-1713953907-aggregator-0      1/1     Running   0          57s
```

4. Configure the settings of the pods for your use case.

    1. Create a file called `my-values.yaml`.
    2. Add the configuration needed for your use case. The settings in this file override the default configuration settings of the chart.
    3. Update your deployment using the `my-values.yaml` file by running:

    ```
    helm upgrade <release-name> axosyslog/axosyslog -f my-values.yaml
    ```

The output should be similar to:

```
Release "axosyslog-1713953907" has been upgraded. Happy Helming!
...
```

> **Tip**
>
> **To list the non-default values of a release, run** `helm get values <release-name>` **.**

Send the collector logs to another destination

By default, the collector sends the logs in JSON format to the aggregator over TCP. To send the logs to a different destination, configure the destination in your values file, then run `helm upgrade`. For example, the following values file sends the logs in JSON format to the `192.0.2.10:514` address over TCP:

```
collector:
  config:
    destinations:
      syslog:
        enabled: true
        transport: tcp
        address: 192.0.2.10
        port: 514
        template: "$(format-json .*)"
```

For details and other parameters, see Collector parameters.

Send test messages to the aggregator

To make sure that the aggregator receives messages, send test messages from the aggregator pod.

1. Run `loggen` in the aggregator pod:

```
kubectl exec <release-name>-aggregator-0 -- loggen -S 127.0.0.1 1514
```

Expected output:

```
count=9328, rate = 882.83 msg/sec
count=9786, rate = 884.20 msg/sec
count=9800, rate = 27.92 msg/sec
average rate = 928.58 msg/sec, count=9800, time=10.5538, (average) msg
size=256, bandwidth=232.14 kB/sec
```

2. Check the configured destinations for the generated messages. For example, with the default file destination:

```
kubectl exec <release-name>-aggregator-0 -- tail -n 5 /var/log/syslog
```

The generated messages look like this:

```
2024-05-02T10:56:31.000000+00:00 localhost prg00000[1234]: seq: 0000000065,
thread: 0000, runid: 1714647391, stamp: 2024-05-02T10:56:31 PADDPADDPADDPADD
```

How to use disk-buffers in containers and Kubernetes

When you are running AxoSyslog in a container or in Kubernetes, and you want to use disk-buffers, there are some additional things to configure.

- Make sure to mount the disk-buffer files and the persist file (by default, both are stored in `/var/lib/syslog-ng`) in a way they are not lost when the pod or container is restarted.
  - In Kubernetes, add a persistent volume to your pod and store the disk buffer files (`/var/lib/syslog-ng`) there.
  - In a container, mount the disk-buffer directory from the host, or store it on a local volume.

- Use a reliable disk-buffer only if your storage is fast enough. For example, a low-speed persistent volume in Kubernetes can cause a significant performance degradation for AxoSyslog.
- Use the latest available version of AxoSyslog, as many related improvements and performance improvements (for example, disk-buffer related metrics) are only supported in recent versions.

If you are using `syslog-ng` without disk-buffering configured, `syslog-ng` stores everything in memory, which results in great performance. If you enable disk-buffering, the performance decreases. Make sure to size your observability pipeline appropriately.

Upgrade the Helm chart

To upgrade the chart to a newer version, and to roll back an upgrade, see Upgrade the Helm chart.

Uninstall the Helm chart

> **Tip**
>
> **To list the installed releases, run `helm list`.**

To uninstall a release of the chart, run:

```
helm uninstall <release-name>
```

4.4.1 Parameters of the AxoSyslog Helm chart

Configurable parameters and default values of the AxoSyslog Helm chart for the collector, aggregator, and metrics exporter.

The following tables list the configurable parameters of the `axosyslog` Helm chart and their default values. For details on installing the chart, see Install AxoSyslog with Helm.

The chart has two components that you can enable or disable independently:

- The collector is a DaemonSet that runs on every node, collects the pod logs, and forwards them to a destination. By default, it forwards the logs to the aggregator.
- The aggregator is a StatefulSet that receives syslog and `axosyslog-otlp()` messages from the network (including the messages of the collector), and routes them to local or remote destinations.

Collector parameters

When you deploy AxoSyslog as a collector (which is a DaemonSet), it collects and forwards local logs to a destination. You can use the following parameters to configure the collector. The parameters for specific destinations are shown in subsequent sections.

Parameter	Description	Default
collector.enabled	Deploy AxoSyslog as a collector to collect and forward local logs.	`true`
collector.config.destinations	The configurations of destinations that can be configured using chart values: syslog , opensearch , and axosyslogOtlp . For destinations and options not available as chart values, you can use the `collector.config.raw` option.	The syslog destination is enabled, and sends the logs to the aggregator.
collector.config.raw	A complete `syslog-ng` configuration. If this parameter is set, all other parameters in the `collector.config` section are ignored. You can use this to set parameters that are not available as chart values. For details on how to create a configuration for `syslog-ng` , see the AxoSyslog documentation .	`""`

collector.config.rewrites.set	A list of name-value pairs to set for the collected log messages. Uses the `set` rewrite rule .	`{}`
collector.config.sources.kubernetes.enabled	Collect pod logs using the `kubernetes()` source. If disabled, the chart doesn't configure any source. For the list of available sources, see the Sources chapter .	`true`
collector.config.sources.kubernetes.prefix	Set JSON prefix for logs collected from the Kubernetes cluster.	`""`
collector.config.sources.kubernetes.keyDelimiter	Set JSON key delimiter for logs collected from the Kubernetes cluster.	`""`
collector.config.sources.kubernetes.maxContainers	The maximum number of containers to collect logs from. Sets the `max-containers()` option of the `kubernetes()` source.	`""`
collector.config.stats.level	Specifies the level of statistics AxoSyslog collects about the processed messages. For details, see level().	`2`

The following example uses the `collector.config.raw` parameter to configure a custom destination:

```
collector:
  config:
    raw: |
      @version: 4.29.0
      @include "scl.conf"

      log {
        source {
          syslog(port(12345));
        };

        destination {
          logscale(
            token("<YOUR_INGEST_TOKEN>")
          );
        };

        flags(flow-control);
      };

  hostNetworking: true
```

Collector syslog destination

Send logs over the network, conforming to RFC3164 using the `network()` destination driver. By default, the collector uses this destination to send the logs to the aggregator in JSON format.

Parameter	Description	Default
collector.config.destinations.syslog.enabled	Enables the destination.	`true`
collector.config.destinations.syslog.address	The IP address or hostname of the destination host. Can include Helm templates.	`<release-name>-aggregator.<namespace>.svc.cluster.local`
collector.config.destinations.syslog.extraOptionsRaw	Other options of the `network()` destination .	`"time-reopen(10)"`

collector.config.destinations.syslog.port	The port number to send the messages to.	`514`
collector.config.destinations.syslog.template	A template to format the messages.	`"$(format-json .*)"`
collector.config.destinations.syslog.transport	The transport protocol to use. Possible values: `tcp` , `udp`	`tcp`

For example, to send the logs to a syslog server outside the cluster:

```
collector:
  config:
    destinations:
      syslog:
        enabled: true
        transport: tcp
        address: 192.0.2.10
        port: 514
        template: "$(format-json .*)"
```

Collector OpenSearch destination

Send logs to OpenSearch over HTTP or HTTPS.

Parameter	Description	Default
collector.config.destinations.opensearch.enabled	Enables the destination.	`false`
collector.config.destinations.opensearch.url	The URL of the OpenSearch server, for example, `http://my-release-opensearch.default.svc.cluster.local:9200` .	`""`
collector.config.destinations.opensearch.index	Name of the OpenSearch index that stores the messages.	`""`
collector.config.destinations.opensearch.user	The username to use for authentication on the OpenSearch server, if not authenticating with a certificate.	`""`
collector.config.destinations.opensearch.password	The password to use for authentication on the OpenSearch server.	`""`

collector.config.destinations.opensearch.template	A template to format the messages.	`"$(format-json .*)"`
collector.config.destinations.opensearch.extraOptionsRaw	Other options of the `elasticsearch-http()` destination.	`"time-reopen(10)"`
collector.config.destinations.opensearch.tls.CADir	A directory containing a set of trusted CA certificates in PEM format. The name of the files must be the 32-bit hash of the subject's name. AxoSyslog verifies the certificate of the server using these CA certificates.	`""`
collector.config.destinations.opensearch.tls.CAFile	The CA certificate in PEM format to use when verifying the certificate of the server.	`""`
collector.config.destinations.opensearch.tls.Cert	Name of a file containing an X.509 certificate or a certificate chain in PEM format. AxoSyslog authenticates with this certificate on the server, with the private key set in the `collector.config.destinations.opensearch.tls.Key` field. 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.	`""`
collector.config.destinations.opensearch.tls.Key	Name of a file containing an unencrypted private key in PEM format. AxoSyslog authenticates with this key and the certificate set in the `collector.config.destinations.opensearch.tls.Cert` field.	`""`
collector.config.destinations.opensearch.tls.peerVerify	If true, AxoSyslog verifies the certificate of the server with the CA certificates set in `collector.config.destinations.opensearch.tls.CAFile` and `collector.config.destinations.opensearch.tls.CADir`.	`false`

For example:

```
collector:
  config:
    destinations:
      syslog:
        enabled: false
      opensearch:
        enabled: true
        url: http://my-release-opensearch.default.svc.cluster.local:9200
        index: "test-axoflow-index"
        tls:
          CAFile: "/path/to/CAFile.pem"
          Cert: "/path/to/Cert.pem"
          Key: "/path/to/Key.pem"
          peerVerify: true
```

Collector axosyslogOtlp destination

Send logs to another AxoSyslog node using the `axosyslog-otlp()` destination driver.

Parameter	Description	Default
collector.config.destinations.axosyslogOtlp.enabled	Enables the destination.	`false`
collector.config.destinations.axosyslogOtlp.url	The IP address or hostname and the port of the destination host. Can include Helm templates.	`<release-name>-aggregator.<namespace>.svc.cluster.local:4317`
collector.config.destinations.axosyslogOtlp.extraOptionsRaw	Other options of the `axosyslog-otlp()` destination .	`"time-reopen(1) batch-timeout(1000) batch-lines(1000)"`

For example, to send the logs to the aggregator using `axosyslog-otlp()` instead of syslog:

```
collector:
  config:
    destinations:
      syslog:
        enabled: false
      axosyslogOtlp:
        enabled: true
```

Other collector parameters

Parameter	Description	Default
collector.affinity	Affinity rules for collector pod scheduling.	`{}`
collector.annotations	Additional annotations for the collector DaemonSet and its pods.	`{}`
collector.extraVolumes	Additional volumes to add to the collector pod.	`[]`
collector.extraVolumeMounts	Additional volume mounts to add to the collector container.	`[]`
collector.hostAliases	Custom entries added to `/etc/hosts` for collector pods.	`[]`
collector.hostNetworking	Use the host network namespace for collector pods.	`false`
collector.labels	Additional labels for the collector DaemonSet and its pods.	`{}`
collector.maxUnavailable	The maximum number of unavailable pods during a rolling update of the DaemonSet.	`1`
collector.nodeSelector	Node selector for collector pod assignment.	`{}`
collector.resources	CPU and memory resource requests and limits for the collector. If not set, the global `resources` value is used.	`{}`
collector.secretMounts	Secrets to mount as files into the collector container.	`[]`

collector.securityContext	Container-level security context for the collector. If not set, the global `securityContext` value is used.	`{}`
collector.tolerations	Tolerations for collector pod scheduling.	`[]`

Aggregator parameters

When you deploy AxoSyslog as an aggregator (which is a StatefulSet), it receives incoming data from the network and routes it to a local or remote destination. You can use the following parameters to configure the aggregator. The parameters for specific sources and destinations are shown in subsequent sections.

Parameter	Description	Default
aggregator.enabled	Deploy AxoSyslog as an aggregator to receive logs from the network.	`true`
aggregator.replicaCount	The number of aggregator replicas.	`1`
aggregator.bufferStorage.enabled	Configures a storage using PersistentVolumes to use as disk-buffer.	`false`
aggregator.bufferStorage.storageClass	The class of the storage to use, for example, `standard`.	`standard`
aggregator.bufferStorage.size	The maximum size of the storage to use as disk-buffer, for example, `10Gi`.	`10Gi`
aggregator.logFileStorage.enabled	Configures a storage using PersistentVolumes to store the log files. The volume is mounted to `/var/log`.	`false`

aggregator.logFileStorage.storageClass	The class of the storage to use, for example, `standard`.	`standard`
aggregator.logFileStorage.size	The maximum size of the storage to use for log storage, for example, `50Gi`.	`50Gi`
aggregator.config.raw	A complete `syslog-ng` configuration. If this parameter is set, all other parameters in the `aggregator.config` section are ignored. You can use this to set parameters that are not available as chart values. For details on how to create a configuration for `syslog-ng`, see the AxoSyslog documentation.	`""`
aggregator.config.stats.level	Specifies the level of statistics AxoSyslog collects about the processed messages. For details, see level().	`2`
aggregator.config.rewrites.set	A list of name-value pairs to set for the received log messages. Uses the `set` rewrite rule.	`{}`
aggregator.config.sources	The configurations of the sources that can be configured using chart values: syslog and axosyslogOtlp. For sources not available as chart values, you can use the `aggregator.config.raw` option.	Both sources are enabled.
aggregator.config.destinations	The configurations of destinations that can be configured using chart values: file, syslog, opensearch, and axosyslogOtlp. For destinations not available as chart values, you can use the `aggregator.config.raw` option.	The file destination is enabled.

Aggregator syslog source

You can use the syslog source to receive RFC3164 or RFC5424 formatted syslog messages. The source uses the `default-network-drivers()` source driver. The following table shows the ports where the aggregator receives the messages:

Traffic	Container port	Service port	NodePort
RFC3164 over UDP	1514	514	30514
RFC3164 over TCP	1514	514	30514
RFC5424 over TCP	1601	601	30601

RFC5424 over TLS (only if `aggregator.config.sources.syslog.tls` is set)	6514	6514	30614

The NodePorts are used only if `service.type` is `NodePort` or `LoadBalancer`. If needed, you can open additional ports using the `service.extraPorts` option.

Parameter	Description	Default
aggregator.config.sources.syslog.enabled	Enable receiving syslog messages.	`true`
aggregator.config.sources.syslog.rfc3164UdpPort	The NodePort for RFC3164-formatted messages over UDP.	`30514`
aggregator.config.sources.syslog.rfc3164TcpPort	The NodePort for RFC3164-formatted messages over TCP.	`30514`
aggregator.config.sources.syslog.rfc5424TcpPort	The NodePort for RFC5424-formatted messages over TCP.	`30601`
aggregator.config.sources.syslog.rfc5424TlsPort	The NodePort for RFC5424-formatted messages over TLS.	`30614`
aggregator.config.sources.syslog.maxConnections	Maximum number of parallel connections.	`1000`
aggregator.config.sources.syslog.initWindowSize	The initial window size used for flow-control .	`100000`
aggregator.config.sources.syslog.tls.peerVerify	If `true`, AxoSyslog requests a certificate from the peers. In this case, you must also set the CA directory or the CA file.	`false`
aggregator.config.sources.syslog.tls.CAFile	A file containing trusted CA certificates. For details, see TLS options .	`""`
aggregator.config.sources.syslog.tls.CADir	The directory for the trusted CA files. For details, see TLS options .	`""`
aggregator.config.sources.syslog.tls.Cert	The certificate file to show to the peer. For details, see TLS options .	`""`
aggregator.config.sources.syslog.tls.Key	The private key file for the certificate. For details, see TLS options .	`""`

Aggregator axosyslogOtlp source

Initializes an `axosyslog-otlp()` source to receive messages from another AxoSyslog node that sends telemetry data using the `axosyslog-otlp()` destination driver. The source receives the messages on port 4317 (container and service port).

Parameter	Description	Default
aggregator.config.sources.axosyslogOtlp.enabled	Enable receiving `axosyslog-otlp()` messages.	`true`
aggregator.config.sources.axosyslogOtlp.port	The NodePort for `axosyslog-otlp()` messages. Used only if `service.type` is `NodePort` or `LoadBalancer`.	`30317`

Aggregator file destination

To write the received logs into files, configure the `aggregator.logFileStorage` and the `aggregator.config.destinations.file` options.

Parameter	Description	Default
aggregator.config.destinations.file.enabled	Enables the file destination.	`true`
aggregator.config.destinations.file.path	The path and filename of the log files. Can include macros. For examples, see file: Store messages in plain-text files.	`"/var/log/syslog"`
aggregator.config.destinations.file.template	The template used to format the log messages. Can include macros.	`""`
aggregator.config.destinations.file.extraOptionsRaw	Other options of the `file()` destination. If the directories used in `aggregator.config.destinations.file.path` do not exist, set `extraOptionsRaw: "create-dirs(yes)"`.	`"create-dirs(yes)"`

For example:

```
aggregator:
  enabled: true
  logFileStorage:
    enabled: true
    storageClass: standard
    size: 50Gi
  config:
    destinations:
      file:
        enabled: true
        path: "/var/log/$HOST/syslog"
        extraOptionsRaw: "create-dirs(yes)"
```

Aggregator OpenSearch destination

Send logs to OpenSearch over HTTP or HTTPS.

Parameter	Description	Default
aggregator.config.destinations.opensearch.enabled	Enables the destination.	`false`
aggregator.config.destinations.opensearch.url	The URL of the OpenSearch server, for example, `http://my-release-opensearch.default.svc.cluster.local:9200`.	`""`
aggregator.config.destinations.opensearch.extraOptionsRaw	Other options of the `elasticsearch-http()` destination.	`"time-reopen(10)"`
aggregator.config.destinations.opensearch.index	Name of the OpenSearch index that stores the messages.	`""`
aggregator.config.destinations.opensearch.user	The username to use for authentication on the OpenSearch server, if not authenticating with a certificate.	`""`
aggregator.config.destinations.opensearch.password	The password to use for authentication on the OpenSearch server.	`""`
aggregator.config.destinations.opensearch.template	A template to format the messages, for example, `"$(format-json --scope rfc5424 --exclude DATE --key ISODATE @timestamp=${ISODATE})"`.	`""`

aggregator.config.destinations.opensearch.tls.CAFile	The CA certificate in PEM format to use when verifying the certificate of the server.	`""`
aggregator.config.destinations.opensearch.tls.CADir	A directory containing a set of trusted CA certificates in PEM format. The name of the files must be the 32-bit hash of the subject's name. AxoSyslog verifies the certificate of the server using these CA certificates.	`""`
aggregator.config.destinations.opensearch.tls.Cert	Name of a file containing an X.509 certificate or a certificate chain in PEM format. AxoSyslog authenticates with this certificate on the server, with the private key set in the `aggregator.config.destinations.opensearch.tls.Key` field. 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.	`""`
aggregator.config.destinations.opensearch.tls.Key	Name of a file containing an unencrypted private key in PEM format. AxoSyslog authenticates with this key and the certificate set in the `aggregator.config.destinations.opensearch.tls.Cert` field.	`""`
aggregator.config.destinations.opensearch.tls.peerVerify	If true, AxoSyslog verifies the certificate of the server with the CA certificates set in `aggregator.config.destinations.opensearch.tls.CAFile` and `aggregator.config.destinations.opensearch.tls.CADir`.	`false`

For example:

```
aggregator:
  enabled: true
  bufferStorage:
    enabled: true
    storageClass: standard
    size: 10Gi
  config:
    destinations:
      opensearch:
        enabled: true
        url: http://my-release-opensearch.default.svc.cluster.local:9200
        index: "test-axoflow-index"
        user: "<YOUR_USERNAME>"
        password: "<YOUR_PASSWORD>"
        #tls:
        #  CAFile: "/path/to/CAFile.pem"
        #  CADir: "/path/to/CADir/"
        #  Cert: "/path/to/Cert.pem"
        #  Key: "/path/to/Key.pem"
        #  peerVerify: false
        extraOptionsRaw: "time-reopen(10)"
```

Aggregator syslog destination

Send logs over the network, conforming to RFC3164 using the `network()` destination driver.

Parameter	Description	Default
aggregator.config.destinations.syslog.enabled	Enables the destination.	`false`
aggregator.config.destinations.syslog.address	The IP address or hostname of the destination host.	`""`
aggregator.config.destinations.syslog.extraOptionsRaw	Other options of the `network()` destination .	`"time-reopen(10)"`
aggregator.config.destinations.syslog.port	The port number to send the messages to.	`""`

aggregator.config.destinations.syslog.template	A template to format the messages.	`""`
aggregator.config.destinations.syslog.transport	The transport protocol to use. Possible values: `tcp`, `udp`	`tcp`

For example:

```
aggregator:
  enabled: true
  bufferStorage:
    enabled: true
    storageClass: standard
    size: 10Gi
  config:
    destinations:
      syslog:
        enabled: true
        transport: tcp
        address: 192.0.2.10
        port: 514
        # convert incoming data to JSON
        #template: "$(format-json .*)\n"
        # use standard syslog logfile
        #template: "$ISODATE $HOST $MSGHDR$MSG\n"
        extraOptionsRaw: "time-reopen(10)"
```

Aggregator axosyslogOtlp destination

Send data using the `axosyslog-otlp()` destination driver to another AxoSyslog node.

Parameter	Description	Default
aggregator.config.destinations.axosyslogOtlp.enabled	Enables the destination.	`false`
aggregator.config.destinations.axosyslogOtlp.url	The IP address or hostname and the port of the destination host. Can include Helm templates.	`""`

aggregator.config.destinations.axosyslogOtlp.extraOptionsRaw	Other options of the `axosyslog-otlp()` destination .	`"time-reopen(1) batch-timeout(1000) batch-lines(1000)"`

For example:

```
aggregator:
  enabled: true
  bufferStorage:
    enabled: true
    storageClass: standard
    size: 10Gi
  config:
    destinations:
      axosyslogOtlp:
        enabled: true
        url: "192.0.2.10:4317"
        extraOptionsRaw: "time-reopen(1) batch-timeout(1000) batch-lines(1000)"
```

Other aggregator parameters

Parameter	Description	Default
aggregator.affinity	Affinity rules for aggregator pod scheduling.	`{}`
aggregator.annotations	Additional annotations for the aggregator StatefulSet and its pods.	`{}`
aggregator.extraVolumes	Additional volumes to add to the aggregator pod.	`[]`
aggregator.extraVolumeMounts	Additional volume mounts to add to the aggregator container.	`[]`
aggregator.hostAliases	Custom entries added to `/etc/hosts` for aggregator pods.	`[]`
aggregator.labels	Additional labels for the aggregator StatefulSet and its pods.	`{}`
aggregator.nodeSelector	Node selector for aggregator pod assignment.	`{}`

aggregator.resources	CPU and memory resource requests and limits for the aggregator. If not set, the global `resources` value is used.	`{}`
aggregator.secretMounts	Secrets to mount as files into the aggregator container.	`[]`
aggregator.securityContext	Container-level security context for the aggregator. If not set, the global `securityContext` value is used.	`{}`
aggregator.tolerations	Tolerations for aggregator pod scheduling.	`[]`

Metrics parameters

You can deploy `axosyslog-metrics-exporter` as a sidecar container of the collector to expose the metrics of AxoSyslog in Prometheus format on port 9577. If you use the Prometheus Operator, you can also deploy a PodMonitor to scrape the metrics.

Parameter	Description	Default
metricsExporter.enabled	Deploy `axosyslog-metrics-exporter` as a sidecar on the collector DaemonSet.	`false`
metricsExporter.image.repository	The image repository of the metrics exporter.	`ghcr.io/axoflow/axosyslog-metrics-exporter`
metricsExporter.image.tag	The image tag of the metrics exporter.	`latest`
metricsExporter.image.pullPolicy	The image pull policy of the metrics exporter. If not set, the default policy of Kubernetes applies.	`""`
metricsExporter.resources	CPU and memory resource requests and limits for the metrics exporter sidecar.	`{}`
metricsExporter.securityContext	Container-level security context for the metrics exporter sidecar.	`{}`
podMonitor.enabled	Deploy a PodMonitor custom resource for the Prometheus Operator. Requires `metricsExporter.enabled`.	`false`
podMonitor.labels	Additional labels for the PodMonitor.	`{}`
podMonitor.annotations	Additional annotations for the PodMonitor.	`{}`

Generic chart parameters

The following parameters apply to both the collector and the aggregator. Where a component has its own parameter with the same name (for example, `collector.resources` or `aggregator.resources`), the component-level setting overrides the generic one.

Parameter	Description	Default
image.repository	The container image repository.	`ghcr.io/axoflow/axosyslog`
image.pullPolicy	The container image pull policy.	`IfNotPresent`
image.tag	The container image tag. If not set, the `appVersion` of the chart is used.	`""`
image.extraArgs	Additional arguments passed to the `syslog-ng` process.	`[]`
imagePullSecrets	The names of secrets containing private registry credentials.	`[]`
nameOverride	Override the chart name.	`""`
fullnameOverride	Override the fully qualified chart name.	`""`
rbac.create	Create a ClusterRole and a ClusterRoleBinding for the collector.	`true`
rbac.extraRules	Additional RBAC rules to add to the ClusterRole.	`[]`
openShift.enabled	Set to `true` when deploying on OpenShift.	`false`
openShift.securityContextConstraints.create	Create SecurityContextConstraints on OpenShift.	`true`
openShift.securityContextConstraints.annotations	Annotations to apply to SecurityContextConstraints.	`{}`
service.create	Create a service so the aggregator can receive incoming connections.	`true`
service.type	The type of the service. Possible values: `NodePort`, `LoadBalancer`, `ClusterIP`, `ExternalName`	`NodePort`
service.annotations	Annotations to apply to the service.	`{}`
service.extraPorts	Additional ports to expose on the service of the aggregator.	`[]`

serviceAccount.create	Create a service account for the pods.	`true`
serviceAccount.annotations	Annotations to apply to the service account.	`{}`
namespace	The Kubernetes namespace to deploy to. If not set, the namespace of the Helm release is used.	`""`
podAnnotations	Annotations applied to all pods.	`{}`
podSecurityContext	Pod-level security context applied to all pods.	`{}`
securityContext	Default container-level security context for all components.	`{}`
resources	Default CPU and memory resource requests and limits for all components.	`{}`
nodeSelector	Default node selector for all pods.	`{}`
tolerations	Default tolerations for all pods.	`[]`
affinity	Default affinity rules for all pods.	`{}`
updateStrategy	Update strategy for the DaemonSet and the StatefulSet.	`RollingUpdate`
priorityClassName	The PriorityClass of the pods.	`""`
dnsConfig	The DNS configuration of the pods .	`{}`
hostAliases	Default entries to the hosts file of the pods .	`[]`
secretMounts	Default secrets to mount as files.	`[]`
extraVolumes	Default additional volumes for all pods.	`[]`
extraVolumeMounts	Default additional volume mounts for all pods.	`[]`
terminationGracePeriodSeconds	The time in seconds given to the pods to terminate gracefully.	`30`

4.5 Install AxoSyslog with Podman

AxoSyslog provides cloud-ready images . These images differ from the upstream `syslog-ng` images, because:

- They're based on Alpine Linux, instead of Debian testing for reliability and smaller size (thus smaller attack surface).
- They incorporate cloud-native features and settings, such as the Kubernetes source.
- They incorporate container-level optimizations for better performance and improved security. For example, they use an alternative malloc library.
- They support the ARM architecture.

The AxoSyslog images support the following architectures:

- amd64
- arm/v7
- arm64

Install the AxoSyslog images

You can find the list of tagged versions at https://github.com/axoflow/axosyslog-docker/pkgs/container/axosyslog .

To install the latest stable version, run:

```
podman pull ghcr.io/axoflow/axosyslog:latest
```

You can also use it as a base image in your Dockerfile:

```
FROM ghcr.io/axoflow/axosyslog:latest
```

If you want to test a development version, you can use the nightly builds:

```
podman pull ghcr.io/axoflow/axosyslog:nightly
```

Note: These named packages are automatically updated when a new package is released. To install a specific version, run `podman pull ghcr.io/axoflow/axosyslog:<version-number>`, for example:

```
podman pull ghcr.io/axoflow/axosyslog:4.29.0
```

Customize the configuration

The AxoSyslog container image stores the configuration file at `/etc/syslog-ng/syslog-ng.conf`. By default, AxoSyslog collects the local system logs and logs received from the network into the `/var/log/messages` and `/var/log/messages-kv.log` files using this configuration file from the syslog-ng repository.

To customize the configuration, create your own configuration file and override the file in the container image with it, for example:

```
podman run --rm --volume <path-to-your/syslog-ng.conf>:/etc/syslog-ng/syslog-ng.conf ghcr.io/axoflow/axosyslog:latest
```

How to use disk-buffers in containers and Kubernetes

When you are running AxoSyslog in a container or in Kubernetes, and you want to use disk-buffers, there are some additional things to configure.

- Make sure to mount the disk-buffer files and the persist file (by default, both are stored in `/var/lib/syslog-ng`) in a way they are not lost when the pod or container is restarted.
  - In Kubernetes, add a persistent volume to your pod and store the disk buffer files (`/var/lib/syslog-ng`) there.
  - In a container, mount the disk-buffer directory from the host, or store it on a local volume.
- Use a reliable disk-buffer only if your storage is fast enough. For example, a low-speed persistent volume in Kubernetes can cause a significant performance degradation for AxoSyslog.

- Use the latest available version of AxoSyslog, as many related improvements and performance improvements (for example, disk-buffer related metrics) are only supported in recent versions.

If you are using `syslog-ng` without disk-buffering configured, `syslog-ng` stores everything in memory, which results in great performance. If you enable disk-buffering, the performance decreases. Make sure to size your observability pipeline appropriately.

Expose port to receive incoming traffic

To receive incoming network in a container, you must expose the port from the container where you want to receive the traffic to the host that's running the container. Typically, this is only needed if you are running AxoSyslog as a relay or a server/aggregator.

By default, the AxoSyslog container images expose the ports commonly used to receive syslog traffic:

- `514/udp`, typically used for RFC3164 (BSD-syslog) formatted traffic.
- `601/tcp`, typically used for RFC5424 (IETF-syslog) formatted traffic.
- `6514/tcp`, typically used for RFC5424 (IETF-syslog) formatted traffic over TLS.

To expose a specific port, use the `--expose` option when starting the container. Make sure to include the IP address of the host to make the port externally accessible.

For example, if you are receiving OpenTelemetry messages using the `opentelemetry()` source, expose the `4317` port:

```
podman run --rm --expose 127.0.0.1:4317:4317/tcp --volume <path-to-your/syslog-ng.conf>:/etc/syslog-ng/syslog-ng.conf ghcr.io/axoflow/axosyslog:latest
```

Using AxoSyslog

After you've installed AxoSyslog, you can configure it just like `syslog-ng`, using the same configurations files (`/etc/syslog-ng/syslog-ng.conf` by default). For details, see the Quick-start guide.

Getting help

If you run into any issues while installing or configuring AxoSyslog, or you have any questions, you can find us on our Discord server .

4.6 Install AxoSyslog with Docker

AxoSyslog provides cloud-ready images . These images differ from the upstream `syslog-ng` images, because:

- They're based on Alpine Linux, instead of Debian testing for reliability and smaller size (thus smaller attack surface).
- They incorporate cloud-native features and settings, such as the Kubernetes source.
- They incorporate container-level optimizations for better performance and improved security. For example, they use an alternative malloc library.
- They support the ARM architecture.

The AxoSyslog images support the following architectures:

- amd64
- arm/v7
- arm64

Install the AxoSyslog images

You can find the list of tagged versions at https://github.com/axoflow/axosyslog-docker/pkgs/container/axosyslog .

To install the latest stable version, run:

```
docker pull ghcr.io/axoflow/axosyslog:latest
```

You can also use it as a base image in your Dockerfile:

```
FROM ghcr.io/axoflow/axosyslog:latest
```

If you want to test a development version, you can use the nightly builds:

```
docker pull ghcr.io/axoflow/axosyslog:nightly
```

Note: These named packages are automatically updated when a new package is released. To install a specific version, run `docker pull ghcr.io/axoflow/axosyslog:<version-number>`, for example:

```
docker pull ghcr.io/axoflow/axosyslog:4.29.0
```

Customize the configuration

The AxoSyslog container image stores the configuration file at `/etc/syslog-ng/syslog-ng.conf`. By default, AxoSyslog collects the local system logs and logs received from the network into the `/var/log/messages` and `/var/log/messages-kv.log` files using this configuration file from the syslog-ng repository.

To customize the configuration, create your own configuration file and override the file in the container image with it, for example:

```
docker run --rm --volume <path-to-your/syslog-ng.conf>:/etc/syslog-ng/syslog-ng.conf ghcr.io/axoflow/axosyslog:latest
```

How to use disk-buffers in containers and Kubernetes

When you are running AxoSyslog in a container or in Kubernetes, and you want to use disk-buffers, there are some additional things to configure.

- Make sure to mount the disk-buffer files and the persist file (by default, both are stored in `/var/lib/syslog-ng`) in a way they are not lost when the pod or container is restarted.
  - In Kubernetes, add a persistent volume to your pod and store the disk buffer files (`/var/lib/syslog-ng`) there.
  - In a container, mount the disk-buffer directory from the host, or store it on a local volume.
- Use a reliable disk-buffer only if your storage is fast enough. For example, a low-speed persistent volume in Kubernetes can cause a significant performance degradation for AxoSyslog.

- Use the latest available version of AxoSyslog, as many related improvements and performance improvements (for example, disk-buffer related metrics) are only supported in recent versions.

If you are using `syslog-ng` without disk-buffering configured, `syslog-ng` stores everything in memory, which results in great performance. If you enable disk-buffering, the performance decreases. Make sure to size your observability pipeline appropriately.

Expose port to receive incoming traffic

To receive incoming network in a container, you must expose the port from the container where you want to receive the traffic to the host that's running the container. Typically, this is only needed if you are running AxoSyslog as a relay or a server/aggregator.

By default, the AxoSyslog container images expose the ports commonly used to receive syslog traffic:

- `514/udp`, typically used for RFC3164 (BSD-syslog) formatted traffic.
- `601/tcp`, typically used for RFC5424 (IETF-syslog) formatted traffic.
- `6514/tcp`, typically used for RFC5424 (IETF-syslog) formatted traffic over TLS.

To expose a specific port, use the `--expose` option when starting the container. Make sure to include the IP address of the host to make the port externally accessible.

For example, if you are receiving OpenTelemetry messages using the `opentelemetry()` source, expose the `4317` port:

```
docker run --rm --expose 127.0.0.1:4317:4317/tcp --volume <path-to-your/syslog-ng.conf>:/etc/syslog-ng/syslog-ng.conf ghcr.io/axoflow/axosyslog:latest
```

Using AxoSyslog

After you've installed AxoSyslog, you can configure it just like `syslog-ng`, using the same configurations files (`/etc/syslog-ng/syslog-ng.conf` by default). For details, see the Quick-start guide.

Getting help

If you run into any issues while installing or configuring AxoSyslog, or you have any questions, you can find us on our Discord server .

4.7 AxoSyslog on macOS

Build AxoSyslog from source on macOS with Homebrew dependencies, and see which modules the macOS build leaves out.

AxoSyslog has no official macOS package or Homebrew formula. For production use, run AxoSyslog on a Linux host or as a container. For details, see Install AxoSyslog and Install AxoSyslog with Docker.

To develop, test, or collect logs locally on a Mac, build AxoSyslog from source. The upstream CI builds and tests AxoSyslog on macOS 15 and the latest macOS GitHub runner.

> **Note**
>
> **The macOS CI workflow is the authoritative source for the exact environment variables and build flags. The steps on this page are an outline, and the flags can change between releases.**

Build from source

1. Install Homebrew , then clone the source code.

```
git clone https://github.com/axoflow/axosyslog.git
cd axosyslog
```

2. Install the build dependencies listed in the `contrib/Brewfile` .

```
brew update
brew bundle --file=contrib/Brewfile
```

3. Point the build to the Homebrew libraries and to the Homebrew version of `bison` .

```
HOMEBREW_PREFIX="$(brew --prefix)"
export
PKG_CONFIG_PATH="${HOMEBREW_PREFIX}/opt/openssl@3/lib/pkgconfig:${HOMEBREW_PREF
IX}/opt/net-
snmp/lib/pkgconfig:${HOMEBREW_PREFIX}/lib/pkgconfig:${PKG_CONFIG_PATH}"
export PATH="${HOMEBREW_PREFIX}/opt/bison/bin:${PATH}"
export CFLAGS="-I${HOMEBREW_PREFIX}/include"
export LDFLAGS="-L${HOMEBREW_PREFIX}/lib"
```

4. Configure, build, and install AxoSyslog. Replace `<install-dir>` with the directory to install to. You can use either autotools or CMake.

    - With autotools:

    ```
    ./autogen.sh
    ./configure --prefix=<install-dir> \
      --with-ivykis=system --with-python=3 --with-systemd-journal=no \
      --disable-smtp --disable-java --disable-java-modules \
      --disable-pacct --disable-stackdump --disable-jit
    make
    make install
    ```

    - With CMake:

    ```
    cmake --install-prefix <install-dir> -B build . \
      -DIVYKIS_SOURCE=system -DPYTHON_VERSION=3 -DENABLE_JOURNALD=OFF \
      -DENABLE_AFSMTP=OFF -DENABLE_GRPC=OFF -DENABLE_JAVA=OFF \
      -DENABLE_JAVA_MODULES=OFF -DENABLE_PACCT=OFF -DENABLE_LIBUNWIND=OFF
    cmake --build build --target install
    ```

Limitations

The macOS builds disable the following modules:

- the `systemd-journal()` source
- the `smtp()` destination

- Java and the Java-based modules
- the `pacct()` source
- stack dumps on crash (`--disable-stackdump`)

The CMake build also disables the gRPC-based modules, for example, `opentelemetry()` , `loki()` , and `bigquery()` . On macOS 14, an incompatibility between Apache Arrow and Apple clang (apache/arrow#49841) breaks the `arrow-flight` module. Disable it with `--disable-arrow-flight` (autotools) or `-DENABLE_ARROW_FLIGHT=OFF` (CMake).

Collect macOS system logs

macOS has its own logging system. To collect its logs, use the `darwin-oslog()` and `darwin-oslog-stream()` sources. For details, see Collect native macOS system logs.

Run as a service

The source tree includes a sample `launchd` property list at `contrib/com.syslog-ng.syslog-ng.plist` . Adjust the paths to match your `<install-dir>` before you use it.

4.8 Upgrade AxoSyslog to a newer version

Upgrade an existing AxoSyslog installation from packages, container images, or the Helm chart, update the configuration version, and roll back if needed.

Use this page if you already run AxoSyslog and you want to move to a newer release. To replace the `syslog-ng` packages of your distribution with AxoSyslog, see Upgrade syslog-ng to AxoSyslog instead.

An upgrade has two independent parts:

1. Upgrade the AxoSyslog binaries: the packages, the container image, or the Helm chart.
2. Update the `@version:` line of your configuration file. This step is optional. Until you do it, AxoSyslog keeps the behavior of the declared version. For details, see Update the configuration version .

Before you upgrade AxoSyslog

1. Check which version you run now. Write it down, so that you can roll back to it if necessary.

   ```
   syslog-ng --version
   ```

   The first line of the output shows the version of the binary, for example:

   ```
   axosyslog 4 (4.10.1)
   Config version: 4.2
   Installer-Version: 4.10.1
   ```

2. Read the changes of every release between your current version and the target version. Don't skip the intermediate releases.
   - In What's new, look for deprecated options and for **Breaking change** subsections (for example, in Version 4.16).
   - The AxoSyslog release notes on GitHub list every change of every release, including bugfixes.
3. Back up your configuration and your state files:

- `/etc/syslog-ng/` : the configuration files, including the files that you include from the main configuration file.
- `/var/lib/syslog-ng/` : the persist file (`syslog-ng.persist`) and the disk-buffer files. If you store these files in a different location (for example, with the `--persist-file` command-line option, or the `dir()` option of a disk-buffer), back up that location too.

For example:

```
sudo tar -czf axosyslog-backup-$(date +%Y%m%d).tar.gz /etc/syslog-ng
/var/lib/syslog-ng
```

For containers, back up the host directories that you mount into the container. For example, with the default Podman with systemd setup, these are `/opt/axosyslog/etc` and `/var/lib/syslog-ng` .

Upgrade the AxoSyslog packages

The packages keep your existing configuration file (`/etc/syslog-ng/syslog-ng.conf`). Upgrade the module packages together with the base package: the installed `axosyslog-*` module packages must have the same version as the base package.

1. Upgrade the AxoSyslog packages.
   - Debian and Ubuntu:
     1. Update the package lists.

        ```
        sudo apt update
        ```

     2. Upgrade every installed AxoSyslog package.

        ```
        sudo apt install --only-upgrade axosyslog*
        ```

     To upgrade all the packages on the host, run `sudo apt upgrade` instead of the previous step.
   - RHEL, Fedora, and AlmaLinux:

1. Update the package lists.

```
sudo dnf update
```

2. Upgrade every installed AxoSyslog package.

```
sudo dnf upgrade 'axosyslog*'
```

This command upgrades the base package and every installed module package. To upgrade all the packages on the host, run `sudo dnf upgrade` instead of the previous step.

2. Restart the service to start the new binary.

```
sudo systemctl restart syslog-ng
```

3. Check that the service runs.

```
sudo systemctl status syslog-ng
```

Make sure that the `Active:` line of the output shows `active (running)`.

4. Check the version of the binary.

```
syslog-ng --version
```

Make sure that the first line of the output shows the new version.

5. Check the log of AxoSyslog for warnings about your configuration file. For example:

```
sudo journalctl -u syslog-ng -b | grep -i warning
```

If your configuration declares an older version, you see warnings about compatibility mode and about incompatible changes. AxoSyslog works in this state. To resolve the warnings, see Update the configuration version .

Upgrade containers

The AxoSyslog container images are available at `ghcr.io/axoflow/axosyslog`. The `latest` tag moves to every new release automatically. For production, pin a specific version, such as `ghcr.io/axoflow/axosyslog:4.29.0`. Then you decide when to upgrade, and you can roll back to a known tag. For the available tags, see the list of image tags .

Upgrade a Docker or Podman container

If you use Podman, replace `docker` with `podman` in the commands of this section.

1. Pull the new image. Replace `<new-version>` with the version number, such as `4.29.0`.

```
docker pull ghcr.io/axoflow/axosyslog:<new-version>
```

2. Check your configuration file with the new image. The entrypoint of the image is `/usr/sbin/syslog-ng -F`, so you can add the `--syntax-only` option to the end of the command. Mount the directory that contains your configuration file, so that the check also finds the files that you include from it:

```
docker run --rm --volume <path-to-your/config-directory>:/etc/syslog-ng
ghcr.io/axoflow/axosyslog:<new-version> --syntax-only
```

If the configuration is valid, the command doesn't print errors. Check the output for warnings about incompatible changes.

3. Stop and remove the running container.

```
docker stop <container-name>
docker rm <container-name>
```

4. Start a new container from the new image. Use the same options, port mappings, and volume mounts that you used for the old container, and change only the image tag. For example:

   ```
   docker run -d --name <container-name> -p 514:514/udp -p 601:601/tcp --volume <path-to-your/config-directory>:/etc/syslog-ng --volume <path-to-your/persist-directory>:/var/lib/syslog-ng ghcr.io/axoflow/axosyslog:<new-version>
   ```

   Replace the `-p` options with the ports that your old container published. Mount the same `/var/lib/syslog-ng` directory as before. It stores the persist file and the disk-buffer files. For details, see Install AxoSyslog with Docker and Install AxoSyslog with Podman.

5. Check the logs of the new container.

   ```
   docker logs <container-name>
   ```

   The startup message shows the new version, for example: `syslog-ng starting up; version='4.29.0'` .

Upgrade a Podman systemd service

The `AXOSYSLOG_IMAGE` environment variable sets the image of the Podman systemd service. The default value in `/etc/containers/systemd/axosyslog.container` is:

```
Environment="AXOSYSLOG_IMAGE=ghcr.io/axoflow/axosyslog:latest"
```

1. Set the new image tag.

   - If you use a pinned version, change the tag in `/etc/containers/systemd/axosyslog.container` . Alternatively, run `sudo systemctl edit axosyslog` , and set the variable in the override file:

     ```
     [Service]
     Environment="AXOSYSLOG_IMAGE=ghcr.io/axoflow/axosyslog:<new-version>"
     ```

- If you use the `latest` tag, pull the new image. Podman doesn't pull a new `latest` image if one is already available locally.

```
sudo podman pull ghcr.io/axoflow/axosyslog:latest
```

2. Reload the systemd configuration, and restart the service.

```
sudo systemctl daemon-reload
sudo systemctl restart axosyslog
```

3. Check the log of the service.

```
journalctl -b -u axosyslog | tail -100
```

The `syslog-ng starting up` message shows the new version. For details, see Install AxoSyslog with Podman and systemd.

Upgrade the Helm chart

The chart version and the AxoSyslog version are different. Every chart version has an `appVersion`, and the chart uses the image with that tag by default. If you set the `image.tag` parameter, the chart uses that image tag, independently of the chart version. For the list of parameters, see Parameters of the AxoSyslog Helm chart.

> **CAUTION:**
>
> If you don't set the `collector.config.raw` or `aggregator.config.raw` parameters, the chart generates the configuration file. The generated `@version:` is the major and minor version of the `appVersion` of the chart. A chart upgrade therefore changes the configuration version, and turns on the new default behavior of that release. Read What's new before you upgrade the chart.
>
> If you use `config.raw`, you control the `@version:` line of the configuration. The chart doesn't change it.

1. Update the chart repository.

```
helm repo update
```

2. List the available chart versions. The `APP VERSION` column shows the AxoSyslog version of each chart version.

```
helm search repo axosyslog/axosyslog --versions
```

3. Upgrade the release. Use your values file, so that you keep your settings. To install a specific chart version, add the `--version <chart-version>` option.

```
helm upgrade <release-name> axosyslog/axosyslog -f my-values.yaml
```

   The output should be similar to:

```
Release "<release-name>" has been upgraded. Happy Helming!
...
```

   The collector DaemonSet and the aggregator StatefulSet use the `RollingUpdate` update strategy, so Kubernetes replaces the pods one by one. For the collector, the `collector.maxUnavailable` parameter sets how many pods can be unavailable during the update (default: `1`).

4. Check that the new pods run.

```
kubectl get pods
```

5. Check the revision history of the release. You need the revision number to roll back.

```
helm history <release-name>
```

For details on the chart, see Install AxoSyslog with Helm.

Update the configuration version

The `@version:` line of the configuration file declares which version of the configuration syntax and default behavior you use. For example:

```
@version: 4.29
@include "scl.conf"
```

The `@version:` line must be the first line of the main configuration file, before any `@include` statement. If the configuration contains more than one `@version:` line, AxoSyslog uses only the first one. For details, see The configuration syntax in detail.

AxoSyslog handles the declared version as follows:

- If you set `@version: current`, AxoSyslog uses the version of the installed binary. This is convenient, but every upgrade can change the behavior of your configuration, without a change in the configuration file.
- If the declared version is older than the version in the `Config version:` line of `syslog-ng --version`, AxoSyslog runs in compatibility mode. It keeps the old default behavior, and logs the `Configuration file format is too old, syslog-ng is running in compatibility mode` warning. It also logs a warning for each version-dependent default that your configuration uses.
- If the declared version is newer than the version of the binary, AxoSyslog logs a warning, and runs with the newest version that it supports.

To update the configuration version, complete the following steps.

1. Upgrade the binary first. Keep the old `@version:` line for now.
2. Check the configuration, and read the warnings.

   ```
   sudo syslog-ng --syntax-only
   ```

   You can also find the warnings in the log of AxoSyslog after a restart.
3. Resolve every warning about incompatible changes. For each warning, do one of these: set the option explicitly to keep the old behavior, or accept the new default behavior.
4. Change the `@version:` line to the new version, for example:

```
@version: 4.29
```

5. Check the configuration again.

```
sudo syslog-ng --syntax-only
```

6. Reload the configuration.

```
sudo syslog-ng-ctl reload
```

   Alternatively, restart the service: `sudo systemctl restart syslog-ng` .

Roll back an AxoSyslog upgrade

If the new version doesn't work as you expect, go back to the version that you wrote down in Before you upgrade AxoSyslog .

If you changed the `@version:` line of the configuration, restore the configuration from your backup. An older binary can't use a newer configuration version. It uses its own newest version instead, so the behavior can change.

Roll back the packages

- Debian and Ubuntu:

  1. List the available versions of the package.

     ```
     apt-cache policy axosyslog
     ```

     You can also run `apt list -a axosyslog` .

  2. Install the previous version. Install the same version of every module package that you use. Replace `<package-version>` with a version from the output of the previous step. For example:

```
sudo apt install --allow-downgrades axosyslog=<package-version> axosyslog-core=<package-version> axosyslog-mod-grpc=<package-version>
```

- RHEL, Fedora, and AlmaLinux:
  - To go back to the previous available version, run:

```
sudo dnf downgrade 'axosyslog*'
```

  - To install a specific version:
    1. List the installed AxoSyslog packages.

```
dnf list --installed 'axosyslog*'
```

    2. Install the previous version of every package from the list. Replace `<version>` with the version to roll back to, for example:

```
sudo dnf install axosyslog-<version> axosyslog-grpc-<version>
```

  - To undo the upgrade transaction, find its ID with `dnf history`, then run:

```
sudo dnf history undo <transaction-id>
```

After the downgrade, restart the service, and check the version:

```
sudo systemctl restart syslog-ng
syslog-ng --version
```

Roll back containers

Start the container again with the previous image tag. Use the same options and volume mounts as for the new container. For Podman with systemd, set the previous tag in the `AXOSYSLOG_IMAGE` variable, then run `sudo systemctl daemon-reload` and `sudo systemctl restart axosyslog`.

If you used the `latest` tag before the upgrade, `latest` now points to the new release. Use the version-number tag of the old version, for example, `ghcr.io/axoflow/axosyslog:<old-version>`. You wrote down this version in Before you upgrade AxoSyslog.

Roll back the Helm chart

1. List the revisions of the release.

```
helm history <release-name>
```

2. Roll back to the revision before the upgrade.

```
helm rollback <release-name> <revision>
```

Get help with an upgrade

If AxoSyslog doesn't start after an upgrade or a rollback, keep the backup of `/etc/syslog-ng/` and `/var/lib/syslog-ng/`, and contact us.

4.9 Upgrade syslog-ng to AxoSyslog

If you're already using `syslog-ng`, you can upgrade your existing `syslog-ng` deployments to AxoSyslog in a matter of minutes, by simply installing AxoSyslog on the host.

If you already run AxoSyslog and want to upgrade it to a newer version, see Upgrade AxoSyslog to a newer version.

We assume that you've installed `syslog-ng` from the repositories of your distribution. To upgrade to AxoSyslog, complete the following steps.

1. Check that the syslog-ng service is running:

```
sudo systemctl status syslog-ng
```

   The output will look something like:

```
syslog-ng.service - System Logger Daemon
 Loaded: loaded (/usr/lib/systemd/system/syslog-ng.service; enabled; preset>
 Active: active (running) since Thu 2025-02-27 17:04:28 CET; 11s ago
   Docs: man:syslog-ng(8)
Main PID: 254 (syslog-ng)
  Tasks: 2 (limit: 9594)
 Memory: 19.6M (peak: 20.8M)
    CPU: 215ms
 CGroup: /system.slice/syslog-ng.service
         └─254 "[rosetta]" /usr/sbin/syslog-ng /usr/sbin/syslog-ng -F
```

2. Check the version of `syslog-ng` you have by running:

```
syslog-ng --version
```

   The output will start with something like:

```
syslog-ng 4 (4.8.1)
Config version: 4.2
Installer-Version: 4.8.1
```

3. Add the AxoSyslog repository for your distribution and install AxoSyslog. For details, see the installation sections:

   - Install AxoSyslog on Debian/Ubuntu
   - Install AxoSyslog on RHEL/Fedora/AlmaLinux

   The installation replaces the `syslog-ng` packages with AxoSyslog packages, but provides the same binaries (for example, `/usr/sbin/syslog-ng`). It will keep using the existing configuration files (`/etc/syslog-ng/syslog-ng.conf`), certificates, and so on.

4. Check that the `syslog-ng` service is still running:

   ```
   sudo systemctl status syslog-ng
   ```

   The output should be identical to the earlier result:

   ```
   syslog-ng.service - System Logger Daemon
    Loaded: loaded (/usr/lib/systemd/system/syslog-ng.service; enabled; preset>
    Active: active (running) since Thu 2025-02-27 17:07:41 CET; 42s ago
      Docs: man:syslog-ng(8)
   Main PID: 1936 (syslog-ng)
     Tasks: 2 (limit: 9594)
    Memory: 79.9M (peak: 83.1M)
       CPU: 462ms
    CGroup: /system.slice/syslog-ng.service
            └─1936 "[rosetta]" /usr/sbin/syslog-ng /usr/sbin/syslog-ng -F
   ```

5. Check the version of `syslog-ng` you have by running:

   ```
   syslog-ng --version
   ```

   You'll see that now you're running AxoSyslog:

   ```
   axosyslog 4 (4.10.1)
   Config version: 4.2
   Installer-Version: 4.10.1
   ```

In case you run into any issues, contact us .

4.10 Forward Windows logs

Axoflow provides a custom OpenTelemetry Collector distribution that you can use to collect logs on Windows hosts and forward them to AxoSyslog using the OpenTelemetry Protocol (OTLP/gRPC).

The distribution provides installers for AMD64 and ARM64 based Windows for:

- Windows Server 2025
- Windows Server 2022
- Windows Server 2019
- Windows 11

Steps

To forward Windows logs to AxoSyslog, complete the following steps.

1. Configure an `opentelemetry()` source on the AxoSyslog that will receive the Windows logs.
2. Download the installation package for your platform (https://github.com/axoflow/axoflow-otel-collector-releases/releases/) from the **Assets** section of the Axoflow OpenTelemetry Collector releases page. We provide MSI installers and binary releases for amd64 and arm64 architectures.
3. Run the installer on your Windows host. The installer installs:
   - the collector agent (by default) to `C:\Program Files\Axoflow\OpenTelemetry Collector\axoflow-otel-collector.exe` , and
   - a default configuration file (`C:\ProgramData\Axoflow\OpenTelemetry Collector\config.yaml`) that must be edited before use.
4. Open the configuration file (`C:\ProgramData\Axoflow\OpenTelemetry Collector\config.yaml`).
5. Set the IP address and port of the AxoSyslog host where you want to send data from this Windows host. Use the IP address and port of an `opentelemetry()` source . For example:

```
exporters:
  otlp/axosyslog:
    endpoint: 10.0.2.2:4317
    tls:
      insecure: true
```

Set the TLS settings to match the configuration of the AxoSyslog `opentelemetry()` source .

6. Configure receivers to collect logs of the Windows host, and the pipelines to forward them. For example, to collect event logs from the Application, System, and Security channels:

```
receivers:
  windowseventlog/application:
    channel: application
    raw: true
    suppress_rendering_info: true
  windowseventlog/system:
    channel: system
    raw: true
    suppress_rendering_info: true
  windowseventlog/security:
      channel: security
      raw: true
      suppress_rendering_info: true
service:
  pipelines:
    logs/eventlog:
      receivers: [windowseventlog/application, windowseventlog/system,
windowseventlog/security]
      processors: [resource/agent, resourcedetection/system]
       exporters: [otlp/axosyslog]
```

For details, see the Windows installation Readme and the OpenTelemetry Collector documentation .

7. Save the file.

8. Restart the service.

```
Restart-Service axoflow-otel-collector
```

The agent starts sending data to the configured AxoSyslog.

5 The concepts of AxoSyslog

How AxoSyslog works: message structure and representation, value-pairs, global objects, modes of operation, timezones, and high availability.

This chapter discusses the technical concepts of AxoSyslog.

5.1 The philosophy of AxoSyslog

Typically, AxoSyslog 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 AxoSyslog clients – all run AxoSyslog, and collect the log messages from the various applications, files, and other *sources*. The clients send all important log messages to the remote AxoSyslog server, which sorts and stores them.

5.2 Logging with AxoSyslog

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

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

Sources and destinations are independent objects, *log paths* define what AxoSyslog 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 AxoSyslog is called a *log statement*.

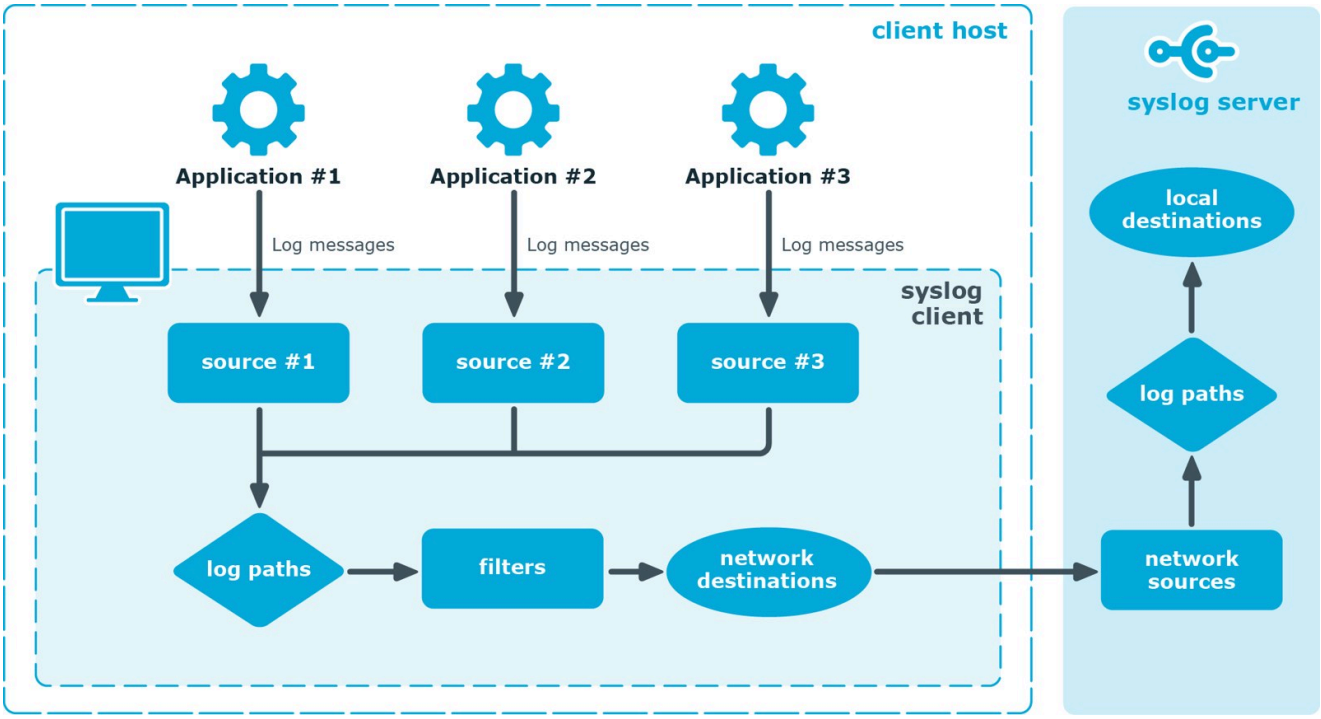
Optionally, log paths can include *filters*. Filters are rules that select only certain messages, for example, selecting only messages sent by a specific application. If a log path includes filters, AxoSyslog sends only the messages satisfying the filter rules to the destinations set in the log path.

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.

5.2.1 The route of a log message in AxoSyslog

Purpose:

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



Steps:

1. A device or application sends a log message to a source on the AxoSyslog client. For example, an Apache web server running on Linux enters a message into the `/var/log/apache` file.
2. The AxoSyslog client running on the web server reads the message from its `/var/log/apache` source.
3. The AxoSyslog client processes the first log statement that includes the `/var/log/apache` source.
4. The AxoSyslog 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, AxoSyslog sends the message to the destinations set in the log statement, for example, to the remote AxoSyslog server.

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

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

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

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

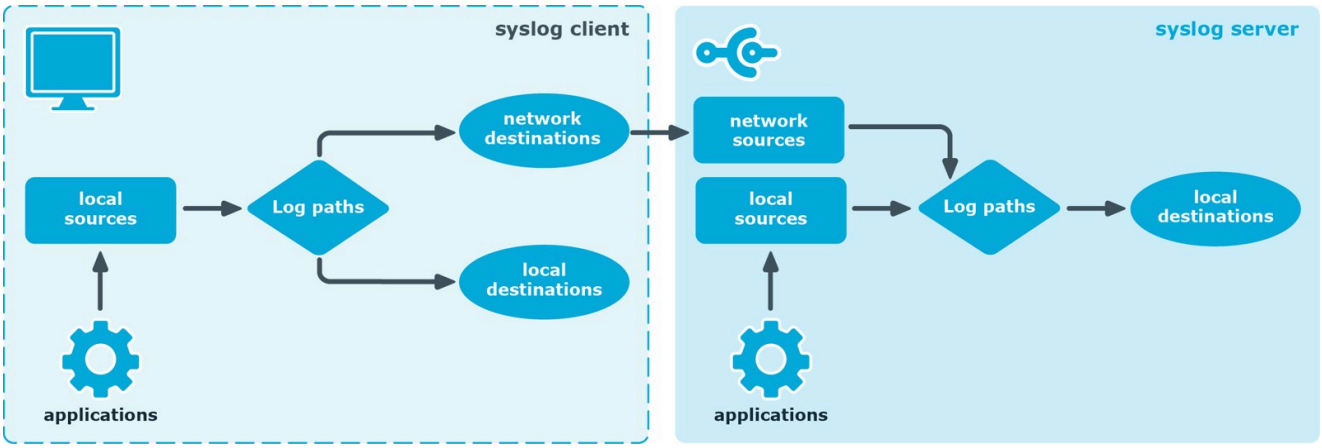
> **Note**
>
> **The AxoSyslog application can stop reading messages from its sources if the destinations cannot process the sent messages. This feature is called flow-control and is detailed in Managing incoming and outgoing messages with flow-control.**

5.3 Modes of operation

The AxoSyslog application has three typical operation scenarios: *Client*, *Server*, and *Relay*.

To set up AxoSyslog in a specific mode, see Quick-start guide.

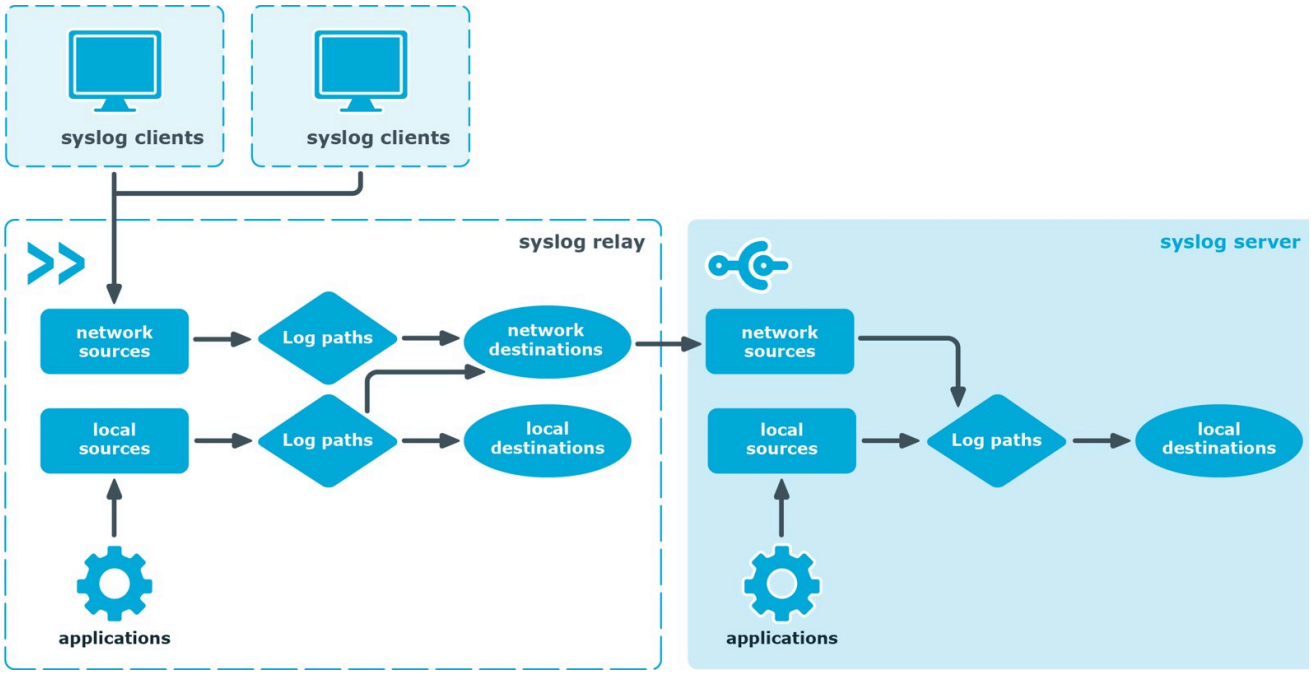
5.3.1 Client mode



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

To configure a client, see Configuring AxoSyslog on client hosts.

5.3.2 Relay mode



In relay mode, AxoSyslog receives logs through the network from AxoSyslog clients and forwards them to the central AxoSyslog 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 AxoSyslog server.

To configure a relay, see Configuring AxoSyslog on relay hosts.

Example relay use cases

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

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

UDP-only source devices

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

To ensure at least a best effort level of reliability, Axoflow recommends that you deploy a relay on the network, close to the source devices. With the most reliable hops between the source and the relay, you can minimize the risk of losing UDP packets. Once the packet arrives at the relay, AxoSyslog ensures that the messages are delivered to the central server in a reliable manner.

Too many source devices

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

If the maximum message rate is lower than 200,000 messages per second:

- maximum ca. 5,000 TCP connections
- maximum ca. 1,000 TLS connections

If the message rate is higher than 200,000 messages per second, contact Axoflow.

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

Collecting logs from remote sites (especially over public WAN)

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

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

Separation, distribution, and balancing of message processing tasks

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

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

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

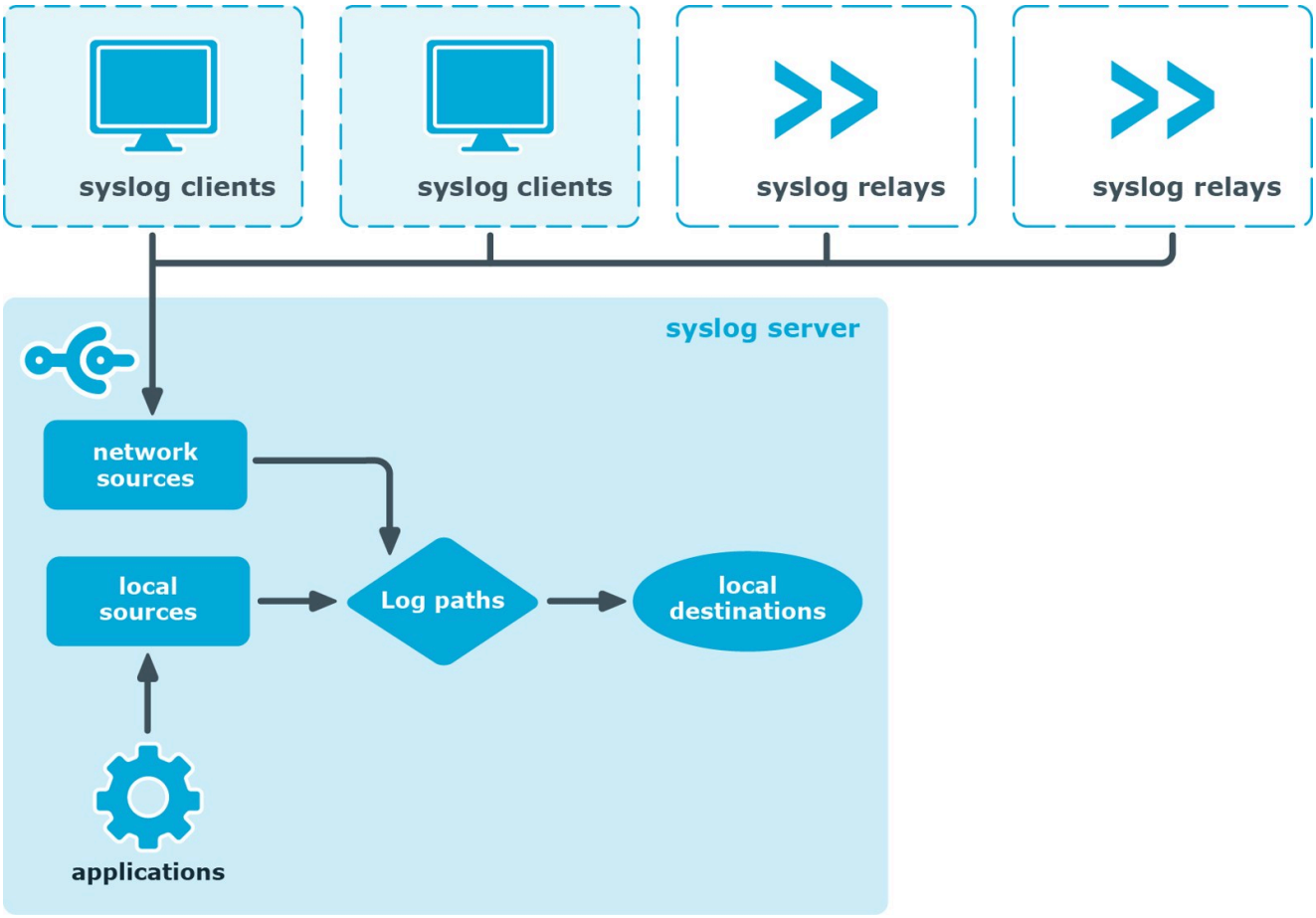
You can run several parallel relays to ensure horizontal redundancy. For example, if each of the relays has the same configuration, when one relay goes down another relay can take over the processing. Distribution of the logs can be done by the built-in client-side failover functionality and also by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

What AxoSyslog relays are not good for

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

If you expect extended outages, Axoflow recommends that you deploy servers instead of relays. There are many deployments where long term storage and archiving are performed on the central AxoSyslog server, but relays also do short-term log storage.

5.3.3 Server mode



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

To configure a server, see Configuring AxoSyslog as a logserver.

5.4 Global objects

The AxoSyslog application uses the following objects:

- *Source driver*: A communication method used to receive log messages. For example, AxoSyslog 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, AxoSyslog 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 Modify messages using rewrite rules.
- *Log paths*: A combination of sources, destinations, and other objects like filters, parsers, and rewrite rules. The AxoSyslog 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.

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

5.5 Timezones and daylight saving

The AxoSyslog 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 AxoSyslog 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 AxoSyslog application also enables administrators to supply timezone information for legacy devices which do not support the protocol extension.

5.5.1 Assigning timezone to the message

When AxoSyslog receives a message, it assigns timezone information to the message using the following algorithm.

1. The sender application 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog application converts the timezone information properly for every individual destination zone.**

> **Warning**
>
> **If AxoSyslog 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 AxoSyslog 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, AxoSyslog uses the local year.)

> **Note**
>
> **You can modify the timezone of the message using timezone-specific rewrite rules. For details, see Rewrite the timezone of a message.**

5.5.2 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 to output timestamps in Unix (POSIX) time format, use the `S_UNIXTIME` and `R_UNIXTIME` macros. You do not need to change any of the timezone related parameters, because the timestamp information of incoming messages is converted to Unix time internally, and Unix time is a timezone-independent time representation. (Actually, Unix time measures the number of seconds elapsed since midnight of Coordinated Universal Time (UTC) January 1, 1970, but does not count leap seconds.)

5.6 Product licensing

Versions 4.12 and newer

Starting with version 4.12, AxoSyslog is licensed under a single, clear open source license: GNU General Public License version 3 or later (GPL-3.0-or-later) . For details on what this means, see the AxoSyslog License Update: Moving to GPL3 blog post.

Versions 3.2 - 4.11

Between versions 3.2 and 4.11, the AxoSyslog application was licensed under a combined LGPL+GPL license. The core of AxoSyslog was licensed under the GNU Lesser General Public License Version 2.1 license, while the rest of the codebase 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.

Documentation license

The Documentation is licensed separately from AxoSyslog. For details, see The syslog-ng OSE Documentation License.

5.7 High availability support

Multiple AxoSyslog servers can be run in fail-over mode. The AxoSyslog application does not include any internal support for this, as clustering support must be implemented on the operating system level.

Starting with AxoSyslog version 3.2, AxoSyslog clients can be configured to send the log messages to failover servers in case the primary syslog server becomes unaccessible. For details on configuring failover servers, see the description of the `failover-servers()` destination option in destination: Forward, send, and store log messages.

5.8 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 AxoSyslog nodes: see Enterprise-wide message model (EWMM)
- How messages are represented in AxoSyslog: see Message representation in AxoSyslog.

Example messages on the wire

The same event in each format, to illustrate how the standards differ:

BSD-syslog (RFC 3164) — the `<PRI>` is followed directly by a legacy `Mmm dd hh:mm:ss` timestamp, the host, and the message:

```
<34>Oct 11 22:14:15 mymachine su: 'su root' failed for lonvick on /dev/pts/8
```

IETF-syslog (RFC 5424) — the `<PRI>` is followed by a version digit and an ISO 8601 timestamp, then the structured header fields:

```
<34>1 2026-10-11T22:14:15+00:00 mymachine su - - - 'su root' failed for lonvick on
/dev/pts/8
```

EWMM — an RFC 5424 frame with `@syslog-ng` as the program and the message carried as JSON:

```
<34>1 2026-10-11T22:14:15+00:00 mymachine @syslog-ng - - -
{"PROGRAM":"su","MESSAGE":"'su root' failed for lonvick on
/dev/pts/8","HOST":"mymachine","._TAGS":[".source.s_network"]}
```

5.8.1 BSD-syslog or legacy-syslog messages

The structure of BSD-syslog (RFC 3164) messages: the PRI, HEADER, and MSG parts.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

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 (the examples are taken from the sample message below):

Part	Example
`PRI`	`<34>`
`HEADER`	`Oct 11 22:14:15 mymachine`
`MSG`	`su: 'su root' failed for lonvick on /dev/pts/8`

The total message cannot be longer than 1024 bytes.

The following is a sample syslog message

```
<34>Oct 11 22:14:15 mymachine su: 'su root' failed for lonvick on /dev/pts/8
```

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**
>
> **RFC 3164 is lenient about what a message must contain: senders may omit the `PRI` (and sometimes the `HEADER`), and receivers still accept such messages. When a message has no `PRI`, AxoSyslog assigns a default facility and severity (`user.notice`, PRI `13`) – see The PRI message part.**

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

The HEADER message part

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

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

> **Note**
>
> **The AxoSyslog application supports other timestamp formats as well, like ISO, or the PIX extended format. For details, see the ts-format() global option.**

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`.

5.8.2 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
```

Part	Example
`PRI`	`<34>`
`VERSION`	`1`
`TIMESTAMP`	`2003-10-11T22:14:15.003Z`
`HOSTNAME`	`mymachine.example.com`
`APP-NAME`	`su`
`PROCID`	`-`
`MSGID`	`ID47`
`STRUCTURED-DATA`	`-`
`MSG`	`'su root' failed for lonvick on /dev/pts/8`

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.

> **Note**
>
> **In RFC 5424, the `PRI` and `VERSION` are mandatory. The remaining `HEADER` fields (`TIMESTAMP` , `HOSTNAME` , `APP-NAME` , `PROCID` , `MSGID`) and the `STRUCTURED-DATA` must be present, but can be the NILVALUE `-` when the sender has no value for them. The `MSG` part is optional.**

The PRI message part

The `PRI` is calculated and interpreted exactly as in BSD-syslog messages (`facility * 8 + severity`). For the formula and the facility and severity code tables, see The PRI message part.

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 AxoSyslog 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 reference.**

The AxoSyslog 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 AxoSyslog application automatically parses the STRUCTURED-DATA part of syslog messages, which can be referenced in macros (for details, see Macros of AxoSyslog). An example STRUCTURED-DATA block looks like:

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

The MSG message part

The MSG part contains the text of the message itself. The encoding of the text must be UTF-8 if the BOM

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

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.

5.8.3 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 AxoSyslog 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 AxoSyslog component in your infrastructure has received it, which is important if you want to forward a message for example, to a SIEM system. To make use of the enterprise-wide message model, you have to use the `syslog-ng()` destination on the sender side , and the `default-network-drivers()` source on the receiver side .

The following is a sample log message in EWMM format.

```
<34>1 2026-10-11T22:14:15+00:00 my-host @syslog-ng - - -
{"PROGRAM":"su","MESSAGE":"'su root' failed for lonvick on
/dev/pts/8","HOST":"mymachine","HOST_FROM":"my-host","._TAGS":
[".source.s_network"]}
```

The message has the following parts:

Part	Example
`HEADER`	`<34>1 2026-10-11T22:14:15+00:00 my-host @syslog-ng - - -`
`MESSAGE`	`{"PROGRAM":"su","MESSAGE":"'su root' failed …","HOST":"mymachine", …, "._TAGS":[".source.s_network"]}`

The header complies with the RFC5424 message format , where the `PROGRAM` field is set to `@syslog-ng` and the `STRUCTURED-DATA` field is empty. The `MESSAGE` part is in JSON format and contains the actual message, as well as any name-value pairs that AxoSyslog has attached to or extracted from the message. The `${._TAGS}` field contains the identifier of the AxoSyslog source that originally received the message on the first AxoSyslog 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 .

5.8.4 The PRI message part

How the PRI part of a syslog message encodes the facility and severity of the message.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

This section describes the `PRI` (priority) part of a syslog message, which encodes the message's facility and severity. The `PRI` is used the same way by the BSD-syslog and IETF-syslog protocols.

The PRI message part

The `PRI` part of the syslog message (known as Priority value) represents the Facility and Severity of the message. Facility represents the part of the system sending the message, while Severity marks its importance.

PRI formula

The Priority value is calculated using the following formula:

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

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

Example: the correlation between facility value, severity value, and the Priority value in the PRI message part

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

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

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

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

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

Facility and Severity values

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

> **Note**
>
> **Facility and severity are set by the sender and used inconsistently, and their names have varied across platforms over time, so a code's name or description is not a reliable indicator of the message. The descriptions follow RFC 5424, while the macro values are the traditional BSD names. AxoSyslog also accepts Facility codes as numerical values.**

syslog Message Facilities

The following table lists possible Facility values. The `Name` column shows the `${FACILITY}` macro value.

Numerical Code (`${FACILITY_NUM}`)	Name (`${FACILITY}`)	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)

syslog Message Severities

The following table lists possible Severity values. The `Name` column shows the `${SEVERITY}` macro value (`${LEVEL}` and `${PRIORITY}` are aliases).

Numerical Code (`${LEVEL_NUM}`)	Name (`${SEVERITY}`)	Severity
0	`emerg`	Emergency: system is unusable
1	`alert`	Alert: action must be taken immediately
2	`crit`	Critical: critical conditions
3	`err`	Error: error conditions

4	`warning`	Warning: warning conditions
5	`notice`	Notice: normal but significant condition
6	`info`	Informational: informational messages
7	`debug`	Debug: debug-level messages

> **Note**
>
> **A message that arrives without a PRI – from a non-syslog source such as OpenTelemetry or a file, or a syslog message that omits the `<PRI>` field – defaults to facility `user` and severity `notice` (PRI `13`). Use the source's `default-facility()` and `default-priority()` options to change this.**

Setting the facility and severity

To override the facility and severity of a message, use the `set-facility()` and `set-severity()` rewrite rules:

```
rewrite r_pri {
    set-facility("local0");
    set-severity("err");
};
```

With FilterX , set the combined PRI value with `set_pri()` (`facility * 8 + severity` , so `local0` (16) and `err` (3) give `131`):

```
filterx {
    set_pri(131);
};
```

Both examples set the message to `local0.err` . Assigning the macros directly (such as `$SEVERITY` or `$PRIORITY`) does not work – they are read-only, macro-based values.

5.9 Message representation in AxoSyslog

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

> **Note**
>
> 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 AxoSyslog. 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 AxoSyslog 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 AxoSyslog. The AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog, 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.

  > **Note**
  >
  > **For the list of hard and soft macros, see Hard versus soft macros.**

Message size and encoding

Internally, AxoSyslog 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, AxoSyslog 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, AxoSyslog splits the message at an undefined location (because the conversion between different encodings is not trivial).

5.10 Structuring macros, metadata, and other value-pairs

Available in AxoSyslog 3.3 and later.

The AxoSyslog 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().

5.10.1 Specifying data types in value-pairs

Prior to version 4.0, AxoSyslog handled every data as strings, and allowed you to convert the strings into other types of data that only certain destinations data formats supported. For example, SQL, MongoDB, JSON, or AMQP support data types like numbers or dates. The AxoSyslog application allows you to specify the data type in templates (this is also called type-hinting or type-casting). If the destination driver supports data types, AxoSyslog 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.

Starting with AxoSyslog 4.0, each name-value pair is a (name, type, value) triplet, and several components of AxoSyslog have typing support, for example, `json-parser()` and the `$(format-json)` template function. For details, see the list of supported data types .

Using explicit type-hinting

You can explicitly type-cast a AxoSyslog template to a specific type. To use type-hinting, enclose the macro or template containing the data with the type: `<datatype>("<macro>")`, for example: `int("$PID")`. See the Type-hinting examples and the list of supported data types for details.

> **Warning**
>
> **Hazard of data loss! If AxoSyslog cannot convert the data into the specified type, an error occurs, and AxoSyslog drops the message by default. To change how AxoSyslog handles data-conversion errors, see Global options .**

Not every destination or other component supports data types. For details, see the list of AxoSyslog components that support data types .

Type-hinting examples

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 following example formats the MESSAGE field as a JSON list.

```
    $(format-json message=list($MESSAGE))"
```

Data types in AxoSyslog

The AxoSyslog 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.
- `json` : A JSON snippet. (Available in AxoSyslog 4.0 and later.)
- `list` : The data as a list. For details, see the list manipulation template functions in Template functions of AxoSyslog.
- `null` : An unset value.
- `integer` : A 32-bit or 64-bit integer, determined by the destination. For example, mongodb uses int32 if the number is less than MAXINT32 and int64 otherwise.
- `string` : The data as a string.

Components that support data types

In AxoSyslog 4.0 and later, the following AxoSyslog components that support data types. Other components treat every data as strings.

- Comparisons in filter expressions: the previously numeric operators are type-aware. The exact comparison depends on the types associated with the values you compare. For details, see Comparing macro values in filters.
- `json-parser()` and the `$(format-json)` template function :

  When using the `json-parser()` , AxoSyslog converts all elements of the JSON object to name-value pairs. Any type information originally present in the incoming JSON object is retained, and automatically propagated to other AxoSyslog components (for example, a destination) if they support types.

  - Elements without a type are treated as strings.
  - JSON lists (arrays) are converted to AxoSyslog lists, so you can manipulate them using the `$(list-*)` template functions .
- `set()` , `groupset()` rewrite rules:

  You can set the type of the field. Where you can use of templates in `set()` and `groupset()` , you can use type-casting, and the type information is properly promoted. For details, see Specifying data types in value-pairs.
- `db-parser()` : The `db-parser()` rules can associate types with values using the `"type"` attribute, for example:

  ```
  <value name="foobar" type="integer">$PID</value>
  ```

  The `integer` is a type-cast that associates `$foobar` with an integer type. `db-parser()` 's internal parsers (for example, `@NUMBER@`) automatically associate type information to the parsed name-value pair.
- `add-contextual-data()` : Name-value pairs that are populated using `add-contextual-data()` propagate type information, similarly to `db-parser()` .
- `map-value-pairs()` : `map-value-pairs()` propagates type information.
- SQL type support: The `sql()` driver supports types, so that columns with specific types are stored as those types.

- Template type support: You can cast templates explicitly to a specific type. Templates also propagate type information from macros, template functions, and values in the template string.
- `python()` typing: All Python components (sources, destinations, parsers, and template functions) support all data types, except `json()`.
- On-disk serialized formats (that is, disk buffer): Version 4.0 and newer are compatible with messages serialized with an earlier version, and the format is compatible for downgrades as well. This means that even if a newer version of AxoSyslog serialized a message, older versions and associated tools are able to read it (but drop the type information of course).

5.10.2 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).**

`value-pairs()` parameters

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 the selection.
Default:	empty string

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

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

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

`omit-empty-values()`

Type:	flag
Default:	N/A

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

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

Available in AxoSyslog version 3.21 and later.

`pair()`

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

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

```
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 AxoSyslog version 3.4.

- `lower`

  Convert all keys to lowercase. Only supports US ASCII.

- `shift("<number>")`

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

- `shift-levels("<number>")`

  Similar to –shift, but instead of cutting characters, it cuts dot-delimited “levels” in the name (including the initial dot). For example, `--shift-levels 2` deletes the prefix up to the second dot in the name of the key: `.iptables.SRC` becomes `SRC`

- `upper`

  Convert all keys to uppercase. Only supports US ASCII.

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 be included in the 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 AxoSyslog 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 AxoSyslog 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 `$SEQNUM` macros.
- *sdata*: The metadata from the structured-data (SDATA) part of RFC5424-formatted messages, that is, every macro that starts with `.SDATA.`
- *everything*: Include every hard and soft macros. This group is mainly useful for debugging, as it contains redundant information (for example, the date-related macros include the date-related information several times in various formats).
- *none*: Reset previously added scopes, for example, to delete automatically-added name-value pairs. The following example deletes every value-pair from the scope, and adds only the ones starting with iptables: `$(format-welf --scope none .iptables.*)`

For example:

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

5.11 Things to consider when forwarding messages between AxoSyslog hosts

When you send your log messages from a AxoSyslog client through the network to a AxoSyslog 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 AxoSyslog you can change many aspects of the network communication. First of all, there is the structure of the messages itself. Currently, AxoSyslog 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 AxoSyslog. 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 AxoSyslog client and the source of your AxoSyslog 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:

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	-	-	!	-	-	!	-	-	✔

Source-destination driver combinations

- `-` 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.

6 The configuration file

Configuration file location and syntax, global variables, modules, inline objects and channels, and ways to manage complex configurations.

6.1 Location of the configuration file

To configure AxoSyslog, edit the `syslog-ng.conf` file with any regular text editor application. The location of the configuration file depends on the platform you are running AxoSyslog, and how you have installed AxoSyslog it.

- **Native packages** of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` directory.
- **Containers**: When running AxoSyslog in a container, typically you map an external file to the `/etc/syslog-ng/syslog-ng.conf` file within the container. Check the mapped volumes of your container, and edit the external file.
- **Kubernetes**: If you're running AxoSyslog in Kubernetes and have installed it with helm, usually you configure AxoSyslog by editing a `values.yaml` file, and redeploying AxoSyslog. Often the `syslog-ng.conf` part is under the `config.raw` section in the `values.yaml` file. For details, see Parameters of the AxoSyslog Helm chart.

6.2 The configuration syntax in detail

Every `syslog-ng.conf` configuration file must begin with a line containing version information. For `syslog-ng` version 4.29, this line looks like:

```
@version: 4.29
```

- If the configuration file does not contain the version information, `syslog-ng` assumes that the file is for 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 can operate with configuration files of version 2.x, but the default values of certain parameters have changed since 3.0.
- `@version: current` sets the configuration version to the currently installed version.

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: 4.29
 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.conf` 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.conf` configuration file are case sensitive.

    - 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.
    - 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 AxoSyslog should use DNS resolution to resolve IP addresses. Global options are detailed in Global options.

```
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));
};
```

> **Note**
>
> 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 using the `/etc/init.d/syslog-ng reload` command.

6.3 Notes about the configuration syntax

When you are editing the `syslog-ng.conf` 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 AxoSyslog 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, AxoSyslog 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.

6.4 Defining configuration objects inline

Starting with AxoSyslog 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");
     };
 };
```

6.5 Using channels in configuration objects

Starting with AxoSyslog 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);
         };
     };
 };
```

6.6 Global and environmental variables

You can 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 (for details, see Regular expressions). To use the variable, insert the name of the variable enclosed between backticks (`` ` `` , similarly to using variables in Linux or UNIX shells) anywhere in the configuration file. If backticks are meant literally, repeat the backticks to escape them. For example:

```
``not-substituted-value``
```

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

- Without any quotes, as long as the value does not contain any spaces or special characters. In other words, 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 (`\\`).

> **Note**
>
> The environmental variables of the host are automatically imported and can be used as global variables.
>
> In AxoSyslog 3.24 and later, the location of the configuration file is available as the `syslog-ng-sysconfdir` variable.

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 AxoSyslog 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));
 };
```

6.7 Configuration identifier

Starting with AxoSyslog version 4.2, you can specify a configuration identifier in the `syslog-ng.conf` file, for example:

```
@config-id: cfg-20230404-13-g02b0850fc
```

This can be useful in managed environments, where AxoSyslog instances and their configuration are automatically deployed or generated.

To show the configuration ID, run `syslog-ng-ctl config --id`

This returns the ID of the currently active configuration, and the SHA256 hash of the configuration (the hash of the output of the `syslog-ng-ctl config --preprocessed` command). The output is similar to:

```
cfg-20230404-13-g02b0850fc
(08ddecfa52a3443b29d5d5aa3e5114e48dd465e195598062da9f5fc5a45d8a83)
```

6.8 Using modules

To increase its flexibility and simplify the development of additional modules, the AxoSyslog application is modular. The majority of AxoSyslog's functionality is in separate modules. As a result, it is also possible to fine-tune the resource requirements of AxoSyslog (for example, by loading only the modules that are actually used in the configuration, or simply omitting modules that are not used but require large amount of memory).

Each module contains one or more plugins that add some functionality to AxoSyslog (for example, a destination or a source driver).

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

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

6.8.1 Loading modules

The AxoSyslog application loads every available module during startup.

To load a module that is not loaded automatically, include the following statement in the AxoSyslog 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. It is usually 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 AxoSyslog configuration file or the external files included in the configuration file:

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

Note

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

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

Note that this is not true for modules marked as mandatory. You can make a dependency module mandatory by defining an error message after the `@requires <module-name>` statement, for example:

```
Example
@requires http "The http() driver is required for elasticsearch-http().
Install syslog-ng-mod-http to continue."
```

6.8.2 Listing configuration options

Starting with AxoSyslog 3.25, you can use the `syslog-ng-cfg-db.py` utility to list the available options of configuration objects. For example, you can list all the options that can be set in the file source, and so on.

The `syslog-ng-cfg-db.py` utility has the following options:

- The following command lists the contexts that the utility supports.

```
syslog-ng-cfg-db.py
```

> **Note**
>
> **Currently, sources and destinations are supported.**

- The following command lists the available drivers of a context:

```
syslog-ng-cfg-db.py -c <source|destination>
```

- The following command lists the available options of a specific driver and specifies the context and the driver:

```
syslog-ng-cfg-db.py -c <source|destination> -d <driver>
```

For example, to list the options of the `kafka-c()` destination driver:

```
syslog-ng-cfg-db.py -c destination -d kafka-c
```

The output includes the available options of the driver in alphabetical order, and the type of the option. For example:

```
destination kafka-c(
    bootstrap-servers/kafka-bootstrap-servers(<string>)
    client-lib-dir(<string>)
    config/option()
    config/option(<string> <arrow> <string-or-number>)
    config/option(<string> <string-or-number>)
    flush-timeout-on-reload(<number>)
    flush-timeout-on-shutdown(<number>)
    frac-digits(<number>)
    key(<string>)
    local-time-zone/time-zone(<string>)
    log-fifo-size(<number>)
    message/template(<string>)
    on-error(<string>)
    persist-name(<string>)
    poll-timeout(<number>)
    properties-file(<path>)
    send-time-zone(<string>)
    sync-send(<yesno>)
    throttle(<number>)
    time-zone(<string>)
    topic(<string>)
    ts-format(<string>)
    workers(<number>)
    config/option(
        <string>(<string-or-number>)
    )
    key(
        <identifier>(<string>)
    )
    message/template(
        <identifier>(<string>)
    )
)
```

> **Note**
>
> **The script caches the list of the options, so if you want to rebuild the database, you have to use the `-r` option.**

6.8.3 Visualize the configuration

Starting with AxoSyslog 3.25, you can visualize the configuration of a running AxoSyslog instance using the `syslog-ng-ctl --export-config-graph` command. The command walks through the effective configuration, and exports it as a graph into a JSON structure.

The resulting JSON file can be converted into DOT file format that visualization tools (for example, Graphviz) can use. The package includes a Python script to convert the exported JSON file into DOT format: `<syslog-ng-installation-directory>/contrib/scripts/config-graph-json-to-dot.py`

You can convert the DOT file into PNG or PDF format using external tools.

6.9 Managing complex configurations

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

6.9.1 Including configuration files

The AxoSyslog 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 AxoSyslog configuration, use the following syntax:

```
@include "<filename>"
```

This imports the entire file into the configuration of AxoSyslog, 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, AxoSyslog will include every matching file. For details on using wildcard characters, see Types and options of regular expressions .
- A directory. When including a directory, AxoSyslog 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:

- The default path where AxoSyslog looks for the file depends on where AxoSyslog is installed. The `syslog-ng --version` command displays this path as `Include-Path` .
- 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 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: 4.29
@include "example.conf"
```

But the following is not:

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

Warning

The AxoSyslog 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.

6.9.2 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. Any AxoSyslog 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`, `options`, `parser`, `rewrite`, `root`, `source`. The `root` blocks can be used in the "root" context of the configuration file, that is, outside any other statements.

Note that `options` can be used in blocks only in version 3.22 and later.

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 AxoSyslog 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.conf` configuration file, reference the block, and use it in a logpath:

```
   @version: 4.29
    @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 AxoSyslog 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);
     };
 };
```

> **Note**
>
> Since the block is inserted into the AxoSyslog configuration when AxoSyslog is started, the block can be generated dynamically using an external script if needed. This is useful when you are running AxoSyslog 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 AxoSyslog 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 AxoSyslog 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) ...) {
        file("`file`" follow-freq(`follow-freq`) format("pacct") tags(".pacct")
`__VARARGS__`);
    };
```

Example: Defining global options in blocks

The following example defines a block called `setup-dns()` to set DNS-related settings at a single place.

```
   block options setup-dns(use-dns()) {
        keep-hostname(no);
        use-dns(`use-dns`);
        use-fqdn(`use-dns`);
    dns-cache(`use-dns`);
    };

    options {
        setup-dns(use-dns(yes));
    };
```

6.9.3 Generating configuration blocks from a script

Purpose:

The AxoSyslog 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 AxoSyslog 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 AxoSyslog configuration snippet.

Steps:

1. Navigate to the directory where you have installed AxoSyslog (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 AxoSyslog 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 AxoSyslog 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.

   You can pass parameters to the script. In the script these parameters are available as environment variables, and have the `confgen_` prefix. For example, passing the `--myparameter` parameter becomes available in the script as the `confgen_myparameter` environment variable.

4. Write a script that generates the output you need, and formats it to a configuration snippet that AxoSyslog 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 AxoSyslog 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 AxoSyslog 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.

6.10 Python code in external files

You can extend and customize AxoSyslog easily by writing destinations , parsers , template functions , and sources in Python.

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

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

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

Note

Starting with 3.26, AxoSyslog assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your AxoSyslog configuration, add a unique `persist-name()` to each source or destination, otherwise AxoSyslog will not start. For example:

```
   log {
        source { python(class(PyNetworkSource) options("port" "8080") persist-
name("<unique-string>); };
        source { python(class(PyNetworkSource) options("port" "8081")); };
    };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name` . For example:

```
   from syslogng import LogSource
      class PyNetworSource(LogSource):
        @staticmethod
        def generate_persist_name(options):
            return options["port"]
        def run(self):
            pass
        def request_exit(self):
            pass
```

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

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

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

6.11 Logging from your Python code

You can extend and customize AxoSyslog easily by writing destinations , parsers , template functions , and sources in Python.

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

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

Logging to the `internal()` source is available in AxoSyslog version 3.20 and later.

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

1. Add the following import to your Python code:

```
import syslogng
```

2. Create a logger object:

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

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

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

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

4. Make sure that your AxoSyslog configuration includes the `internal()` source, for example:

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

7 source: Read, receive, and collect log messages

Configure source drivers to collect messages from files, network protocols, local system logs, OpenTelemetry, Kubernetes, and applications.

7.1 How sources work

A source is where AxoSyslog 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.conf` 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 AxoSyslog 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.

> **Warning**
>
> Sources and destinations are initialized only when they are used in a log statement. For example, AxoSyslog 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:

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");
 };
```

Sources list

Choose a network source

If your clients send...	Use
BSD syslog (RFC3164) over TCP, UDP, or TLS	`network()`
IETF syslog (RFC5424) over TCP, UDP, or TLS	`syslog()`
Mixed syslog traffic that you want to receive and parse automatically	`default-network-drivers()`
OpenTelemetry (OTLP/gRPC) from OpenTelemetry clients	`opentelemetry()`
Logs from another AxoSyslog node	`axosyslog-otlp()`

HTTP or HTTPS requests (webhooks)	`webhook()` or `webhook-json()` , or `ehttp()`
Events for the Splunk HTTP Event Collector (for example, from SC4S)	`splunk-hec()`
Data from Elastic Agent, Beats, or other Elasticsearch Bulk API clients	`elasticsearch-bulk()`

The `default-network-drivers()` source needs `@include "scl.conf"` in your configuration file.

> **Note**
>
> **The `tcp()` , `tcp6()` , `udp()` , and `udp6()` drivers are obsolete. Use `network()` instead. For the migration steps, see Change an old source driver to the network() driver.**

The following table lists the source drivers available in AxoSyslog.

Name	Description
lidarr(), prowlarr(), radarr(), readarr(), sonarr(), whisparr()	Collect logs of *arr media management applications
axosyslog-otlp()	Receive logs from another node using OpenTelemetry
darwin-oslog(), darwin-oslog-stream()	Collect native macOS system logs
default-network-drivers()	Receive and parse common syslog messages
ehttp()	Receive logs over HTTP or HTTPS
elasticsearch-bulk()	Receive messages from clients of the Elasticsearch Bulk API
file()	Collect messages from text files
hypr-audit-trail(), hypr-app-audit-trail()	Fetch events from the Hypr REST API
internal()	Collect internal messages
jellyfin()	Collect Jellyfin logs
kubernetes()	Collect and parse messages in the Kubernetes CRI (Container Runtime Interface) format
linux-audit()	Collect messages from Linux audit logs
mbox()	Convert local email messages to log messages

mqtt()	Fetch messages from MQTT brokers
network()	Collect messages using the RFC3164 protocol
nodejs()	Receive JSON messages from nodejs applications
openbsd()	Collect kernel log messages on OpenBSD systems
opentelemetry()	Receive logs, metrics, and traces from OpenTelemetry clients over the OpenTelemetry Protocol (OTLP/gRPC)
osquery()	Collect and parse osquery result logs
pacct()	Collect process accounting logs on Linux
pihole-ftl()	Collect Pi-hole FTL logs
pipe()	Collect messages from named pipes
program()	Receive messages from external applications
python()	Server-style Python source that receives messages
python-fetcher()	Write a fetcher-style Python source
qbittorrent()	Collect qBittorrent logs
snmptrap()	Read Net-SNMP traps
splunk-hec()	Receive messages sent to the Splunk HTTP Event Collector (HEC)
stdin()	Collect messages from the standard input stream
sun-streams()	Collect messages on Sun Solaris
syslog()	Collect messages using the IETF-syslog protocol
system()	Collect the system-specific log messages of a platform
systemd-journal()	Collect messages from the systemd-journal system log storage
systemd-syslog()	Collect systemd messages using a socket
tcp(), tcp6(), udp(), udp6()	OBSOLETE - Collect messages from remote hosts using the BSD syslog protocol
unix-stream(), unix-dgram()	Collect messages from UNIX domain sockets
webhook(), webhook-json()	Receive logs via a HTTP webhook

wildcard-file()	Collect messages from multiple text files

7.2 Arr logs

AxoSyslog can collect logs of the Lidarr, Prowlarr, Radarr, Readarr, Sonarr, and Whisparr (often referred to as “*Arr” or “*Arrs”) applications.

Use the `*arr()` sources to read various *arr logs:

- `lidarr()`
- `prowlarr()`
- `radarr()`
- `readarr()`
- `sonarr()`
- `whisparr()`

Prerequisites

- AxoSyslog version 4.7.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `radarr()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/arr/arr.conf on GitHub.

Configuration

Example minimal config:

```
source s_radarr {
    radarr(
    dir("/path/to/my/radarr/log/dir")
    );
};
```

The logging module is stored in the `<prefix><module>` name-value pair, for example: `.radarr.module` => `ImportListSyncService`.

You can modify the prefix with the `prefix()` option.

7.3 axosyslog-otlp(): Receive logs from another node using OpenTelemetry

The `axosyslog-otlp()` source and destination allows you to transfer the internal representation of log messages between AxoSyslog instances using the OpenTelemetry protocol. Unlike the traditional AxoSyslog drivers that rely on simple TCP connections, `axosyslog-otlp()` leverages the OpenTelemetry protocol for efficient and reliable log message transmission.

The key benefits of the `axosyslog-otlp()` drivers are:

- scalability (via the `workers()` option),
- built-in application layer acknowledgement,
- support for Google service authentication (ADC or ALTS), and
- improved load balancing capabilities.

To use it, configure an `axosyslog-otlp()` destination on the sender node, and an `axosyslog-otlp()` source on the receiver node, like this:

```
destination d_axosyslog_otlp {
  axosyslog-otlp(url("your-receiver-axosyslog-instance:4317"));
};
```

```
source s_axosyslog_otlp {
  axosyslog-otlp(port(4317));
};
```

Prerequisites

- AxoSyslog version 4.12 and later. (From version 4.4 to 4.11, this driver was called `syslog-ng-otlp()` .)
- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Options

The `axosyslog-otlp()` source has the following options.

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

```
  axosyslog-otlp(
    port(4317)
    auth(alts())
  );
```

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_otlp {
    axosyslog-otlp(
      url("your-otel-server:12346")
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

peer-verify()

Accepted values:	`optional-trusted` , `optional-untrusted` , `required-trusted` , `required-untrusted`
Default:	`required-trusted`

Description: Verification method of the peer, the four possible values is a combination of two properties of validation:

- Whether the peer is required to provide a certificate (required or optional prefix).
- Whether the certificate provided needs to be valid or not.

The following table summarizes the possible options and their results depending on the certificate of the peer.

The remote peer has:		
no certificate	**invalid certificate**	**valid certificate**

Local peer-verify() setting	optional-untrusted	TLS-encryption	TLS-encryption	TLS-encryption
	optional-trusted	TLS-encryption	rejected connection	TLS-encryption
	required-untrusted	rejected connection	TLS-encryption	TLS-encryption

Local peer-verify() setting	required-trusted	reject connection	reject connection	TLS-encryption

For untrusted certificates only the existence of the certificate is checked, but it does not have to be valid – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

channel-args()

Type:	arrow list

Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

concurrent-requests()

Type:	integer
Default:	2

Description: Configures the maximal number of in-flight gRPC requests per worker. Setting this value in the range of 10s or 100s is recommended when there are a high number of clients sending simultaneously. Ideally, `workers() * concurrent-requests()` should be greater than or equal to the number of clients, but this can increase the memory usage.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string

Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

flags()

Type:	`assume-utf8` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`

- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

ip()

Type:	string
Default:	`0.0.0.0`

Available in AxoSyslog 4.23 and later.

Description: The IP address to bind to. By default, AxoSyslog listens on every available interface. Note that this is not the address where messages are accepted from.

keep-alive()

Type:	`yes` or `no`
Default:	`yes`

Available in AxoSyslog 4.20 and later.

Description: Client connections can be kept alive during reload, avoiding unnecessary retry backoffs and other error messages on the client side.

keep-hostname()

The `axosyslog-otlp()` source ignores this option and uses the hostname from the message as the `${HOST}` .

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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()`.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the

`syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

port()

Type:	integer
Default:	`4317`

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`.

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no

Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to process the messages of the source. Increasing the number of worker threads can drastically improve the performance of the source.

7.4 Collect native macOS system logs

AxoSyslog can collect logs on macOS using its native OSLog framework using the `darwin-oslog()` and `darwin-oslog-stream()` source drivers.

- `darwin-oslog()` : This source builds on the native OSLog framework, and replaces the earlier file-source based solution.
- `darwin-oslog-stream()` : Provides a live log stream feed.

Prerequisites

- AxoSyslog version 4.6.0 or later.
- AxoSyslog supports this feature only on macOS. The Debian/Ubuntu and RHEL packages that Axoflow publishes don't include it. To use it, build AxoSyslog from source on macOS.

`darwin-oslog()`

This source is based on the native OSLog Framework to read logs from the local store of the unified logging system on darwin OSes. The AxoSyslog `system()` source automatically uses this new source on darwin platforms if the `darwinosl` plugin is available. This plugin is available only on macOS 10.15 Catalina and above, the first version that has the OSLog API.

Note that the persistent OSLog store usually keeps about 7 days of logs on disk.

The `darwin-oslog()` source has the following options:

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

do-not-use-bookmark()

Type:	boolean
Default:	`no`

Description: By default, AxoSyslog continues to read the logs from the last remembered position after a restart. If this option is set to `yes`, it will always start reading from the end or beginning of the available log list (depending on the setting of the `go-reverse()` option).

fetch-delay()

Type:	integer
Default:	`10000`

Description: Controls the time AxoSyslog waits between reading and sending log messages. This is a fraction of a second, where `wait_time = 1 second / n`, so `n=1` means that only about 1 log is sent in each second, and `n=1000000` means only 1 microsecond is the delay between read/write attempts. The maximal value of this parameter is `1000000`. Note that increasing the value of this parameter (thus lowering delay time) can increase log feed performance, but at the same time could increase system load.

fetch-retry-delay()

Type:	integer
Default:	`1`

Description: Controls how many seconds AxoSyslog waits before trying to check for new logs if there were no more logs to read the last time.

filter-predicate()

Type:	string
Default:	`(eventType == 'logEvent' || eventType == 'lossEvent' || eventType == 'stateEvent' || eventType == 'userActionEvent') && (logType != 'debug')`

Description: String for native macOS log message filtering using predicates. For example, the following predicate selects AirDrop logs: `subsystem=="com.apple.sharing" and category=="AirDrop"`

flags()

Type:	`assume-utf8` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp,

host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

go-reverse()

Type:	boolean
Default:	`no`

Description: Set it to `yes` to process the logs in a reverse order (from latest to oldest).

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

Warning

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

log-fetch-limit()

Type:	integer
Default:	`0` (no limit)

Warning

This option is currently disabled because of an OSLog API bug.

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

CAUTION:

If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-bookmark-distance()

Type:	integer
Default:	`0` (no limit) [seconds]

Description: The maximum distance in seconds that a bookmark can point backwards. That is, if AxoSyslog was stopped for 10 minutes and `max-bookmark-distance()` is set to `60`, then AxoSyslog will start reading the logs from 60 seconds before the startup, losing 9 minutes of logs.

normalize-hostnames()

Accepted values:	`yes`, `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	boolean
Default:	`no`

Description: If set to `yes`, AxoSyslog starts reading logs from the oldest available log when it's first started on a system, or if there are no bookmarks for some reason

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option. The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network

endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

darwin-oslog-stream()

This source is a wrapper around the OS command line `log stream` command that provides a live log stream feed. Unlike the `darwin-oslog()` source, the live stream can contain non-persistent log events as well. This might result in a large number of log events every second.

This source has only one option.

params()

Type:	string
Default:	`--type log --type trace --level info --level debug`

Description: A string that can contain all the possible params the macOS `log` tool can accept. The source uses the "–style" internally (defaults to `ndjson`), so use templates or rewrite rules to format the final output. Use the `def-osl-stream-params` string to reference the default values when extending them with your own.

For a full reference, see the output of the `log --help stream` and `man log` commands.

7.5 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.

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

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

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

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

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

> **Warning**
>
> **If you do not configure the TLS keys to show to the clients, AxoSyslog 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.**

Prerequisites

- AxoSyslog version 3.16.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"`, which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

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

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), AxoSyslog 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 AxoSyslog client using the `syslog-ng()` destination, parse its fields as a AxoSyslog message.

      The Enterprise-wide message model or EWMM allows you to deliver structured messages from the initial receiving AxoSyslog 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 AxoSyslog 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 AxoSyslog (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), AxoSyslog 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 AxoSyslog (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")
         )
     );
 };
```

7.5.1 default-network-drivers() source options

The `default-network-drivers()` driver has the following options.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

flags()

Type:	`assume-utf8` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp,

host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

log-iw-size()

Type:	number
Default:	100

Description: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

max-connections()

Type:	number
Default:	10

Description: Specifies the maximum number of simultaneous connections. Doesn't affect UDP-based traffic, which is always treated as a single connection.

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

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

rfc5424-tcp-port()

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.

Warning

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")
         )
     );
 };
```

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

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.

use-syslogng-pid()

Type:	`yes` , `no`

Default: `no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.6 ehttp: Receive logs over HTTP

The `ehttp()` source receives log messages over HTTP or HTTPS. Clients send the messages in the body of an HTTP request, and AxoSyslog answers every accepted request.

Unlike the `webhook()` source , which is a Python-based configuration snippet, `ehttp()` is implemented natively in AxoSyslog. It is intended to replace `webhook()` in the long run.

> **Warning**
>
> **The `ehttp()` source is experimental. Its name and its options can change in later releases: the driver will be renamed to `http()` once its options are considered stable. Review your configuration when you upgrade AxoSyslog, and do not rely on this source in production yet.**

Prerequisites

- AxoSyslog version 4.28 or later.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Example: Receiving logs over HTTP

The following source accepts JSON payloads on port 8080, requires an authentication token, and answers the requests with a custom response body:

```
source s_http {
  ehttp(
    port(8080)
    mode("json")
    auth-token("Bearer s3cr3t")
    response-body('{"status": "received"}')
    flags(no-parse)
  );
};
```

To test the source, you can use `curl` on the host where AxoSyslog is running:

```
curl -X POST -H "Authorization: Bearer s3cr3t" --data '{"message": "hello"}'
http://127.0.0.1:8080/
```

Note the `flags(no-parse)` option in the example: by default, AxoSyslog parses every extracted message as a syslog message. Use `flags(no-parse)` if the payload is not in syslog format, so that the entire payload becomes the `${MESSAGE}` of the log message.

> **Note**
>
> **The `ehttp()` source accepts requests on every URL path, you cannot restrict it to specific endpoints. If you need path-based routing, use the `webhook()` source and its `paths()` option.**

Compressed requests

The `ehttp()` source accepts `gzip` and `deflate` compressed request bodies, and decompresses them based on the `Content-Encoding` header of the request. The `max-request-size()` limit applies to the decompressed body as well.

Responses and error handling

The `ehttp()` source answers the requests of the clients as follows:

- The request is accepted: `200 OK`, with the body set in `response-body()`, or the `OK` status line if `response-body()` is not set.
- The `auth-token()` option is set, but the `Authorization` header of the request is missing or doesn't match: `401 Unauthorized`
- The request is larger than `max-request-size()`, or its decompressed body is: `413 Payload Too Large`
- The `Content-Encoding` of the request is not supported: `415 Unsupported Media Type`
- The request is not valid HTTP, or its compressed body is corrupt: `400 Bad Request`

If the body of the request cannot be parsed in the configured `mode()`, AxoSyslog logs a warning that includes the parse error and the offset where parsing failed, but still answers the request with `200 OK`.

ehttp() source options

The `ehttp()` source has the following options.

auth-token()

Type:	string
Default:	

Description: Requires the clients to authenticate themselves with a token. When set, AxoSyslog accepts only the requests whose `Authorization` header exactly matches the value of this option, and rejects every other request with `401 Unauthorized`.

Include the authentication scheme in the value, because AxoSyslog compares the whole header:

```
auth-token("Bearer s3cr3t")
```

Since the token is sent in clear text, use this option only over HTTPS, that is, together with `transport(tls)` .

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

flags()

Type:	`assume-utf8` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

format()

Type:	string
Default:	`syslog`

Description: The message format parser that AxoSyslog uses on the extracted messages. In practice the only meaningful value is `syslog`, which is also the default. To skip parsing entirely, use `flags(no-parse)` instead.

host-override()

Type:	string
Default:	

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

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

ip() or localip()

Type:	string
Default:	`0.0.0.0`

Description: The IP address to bind to. By default, AxoSyslog listens on every available interface. Note that this is not the address where messages are accepted from.

ip-freebind()

Type:	yes or no
Default:	no

Description: Linux-only. Enables the `IP_FREEBIND` socket option, allowing AxoSyslog to bind to a non-local IP address. Useful when the address isn't yet configured on any interface (for example, during failover scenarios) so the source can listen before the address becomes available. Has no effect on platforms that don't support `IP_FREEBIND`.

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 AxoSyslog is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the AxoSyslog 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)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.

- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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:	The value of `max-connections()` multiplied by `min-iw-size-per-reader()`

Description: The size of the source window, that is, the maximum number of in-flight messages the source permits before flow control is enforced. This applies only when `flags(flow-control)` is set. For details, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-connections()

Type:	number
Default:	100

Description: The maximum number of simultaneous client connections that the source accepts.

max-request-size()

Type:	number [bytes]
Default:	The value of `log-msg-size()` multiplied by 1000

Description: The maximum size of a single HTTP request, in bytes. AxoSyslog rejects larger requests with `413 Payload Too Large` and logs an error.

For compressed requests, this limit applies to the decompressed body as well.

mode()

Type:	`auto` , `json` , `line-separated` (or `jsonl`), `single`
Default:	`auto`

Description: Determines how AxoSyslog splits the body of the request into log messages.

- `auto` : If the body starts with `{` or `[` and parses as JSON, AxoSyslog handles it as `json` , otherwise as `line-separated` .
- `json` : The body is JSON. If it is an array, every element becomes a separate log message. If it is a sequence of concatenated JSON objects (like the payload of the Splunk HTTP Event Collector), every object becomes a separate log message.
- `line-separated` (or `jsonl`): AxoSyslog splits the body on newline characters, and creates a log message from every non-empty line.
- `single` : The entire body becomes a single log message.

The value is case-insensitive, and must be specified as a string, for example, `mode("json")` .

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

port() or localport()

Type:	number
Default:	80 for `transport(tcp)` , 443 for `transport(tls)`

Description: The port number to bind to. Make sure to enable this port on the firewall of the AxoSyslog host.

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	`yes` or `no`
Default:	yes

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

response-body()

Type:	template or template function
Default:	

Description: The body that AxoSyslog sends in the `200 OK` response. If you don't set this option, the response body is the `OK` status line.

Because the value is a template, you can include the macros of the received message in the response. AxoSyslog evaluates the template using the first log message extracted from the request, for example:

```
response-body('{"status": "received", "host": "${HOST}"}')
```

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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.

> **Warning**
>
> When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the AxoSyslog server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In this cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.
>
> As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.

so-sndbuf()

Type:	number
Default:	0

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

tags()

Type:	string
Default:	

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

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

> **Warning**
>
> 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 AxoSyslog 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.

Warning

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 AxoSyslog 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.

Warning

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 AxoSyslog version 3.4 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.

tls()

Type:	TLS options
Default:	

Description: The TLS settings of the source, for example, the certificate and the private key that AxoSyslog shows to the clients. Setting `transport(tls)` requires a `tls()` block. For details on the individual options, see TLS options.

```
tls(
    key-file("/etc/syslog-ng/server.key")
    cert-file("/etc/syslog-ng/server.crt")
)
```

The following TLS options are available: `ca-dir()`, `cert-file()`, `cipher-suite()`, `crl-dir()`, `dhparam-file()`, `ecdh-curve-list()`, `key-file()`, `peer-verify()`, `pkcs12-file()`, `ssl-options()`, `trusted-dn()`, and `trusted-keys()`.

> **Note**
>
> **This is a narrower set than what the `network()` and `syslog()` sources accept. In particular, `ca-file()` is not available, use `ca-dir()` instead. The `trusted-fingerprints()`, `ocsp-stapling-verify()`, `sigalgs()`, and `keylog-file()` options are not available either.**

transport()

Type:	`tcp` or `tls`
Default:	`tcp`

Description: Sets whether the source accepts plain HTTP (`tcp`) or HTTPS (`tls`) connections. Setting `transport(tls)` requires a `tls()` block, and changes the default port to 443.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.7 elasticsearch-bulk: Receive messages from Elastic clients

The `elasticsearch-bulk()` source implements the Elasticsearch Bulk API . Elastic Agent, Beats, and other clients of the Bulk API can send their events to AxoSyslog by pointing their Elasticsearch output at this source, so you can process and route the events before forwarding them to Elasticsearch or to another destination.

Prerequisites

- AxoSyslog version 4.28 or later.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Example: Receiving events from Beats

```
source s_es {
  elasticsearch-bulk(
    port(9200)
    auth-token("ApiKey <key>")
    version("8.15.0")
  );
};
```

Configure the Elasticsearch output of your client to point at this address, for example, in a Filebeat configuration:

```
output.elasticsearch:
  hosts: ["http://<IP-of-AxoSyslog>:9200"]
  api_key: "<key>"
```

How the messages are processed

The body of a bulk request is a sequence of action lines, each followed by a document line. AxoSyslog handles the actions as follows:

Action	Result
`index` , `create`	The document line becomes a log message, and the action line is stored in `${.es_bulk.action}` . AxoSyslog acknowledges the item with status `201` .
`update` , `delete`	AxoSyslog skips the action and acknowledges the item with status `200` , so the client does not resend it.
An action without a document line	AxoSyslog rejects the item with status `400` , so the client drops a pair it could never parse instead of resending it.

Because bulk documents are JSON and not syslog messages, this source doesn't parse the documents as syslog. The whole document line becomes the `${MESSAGE}` of the log message.

AxoSyslog treats a request as a bulk request if its method is `POST` or `PUT` , and its path ends with `/_bulk` . It answers the requests of the clients as follows:

Request	Response
A bulk request	`200 OK` , with a bulk acknowledgment body that lists the status of every item.
`GET /_license`	The license response that the Elastic clients expect.
Any other `GET` request	The version handshake, which reports the version set in `version()` .
Any other request	`200 OK`
The `auth-token()` option is set, and the `Authorization` header of the request is missing or doesn't match.	`401 Unauthorized`

Every response includes the `X-Elastic-Product: Elasticsearch` header, which the Elastic clients require.

The source accepts `gzip` and `deflate` compressed request bodies, and the `max-request-size()` limit applies to the decompressed body as well.

elasticsearch-bulk() source options

The `elasticsearch-bulk()` source has the following options.

auth-token()

Type:	string
Default:	

Description: Requires the clients to authenticate themselves with a token. When set, AxoSyslog accepts only the requests whose `Authorization` header exactly matches the value of this option, and rejects every other request with `401 Unauthorized`.

Include the authentication scheme in the value, because AxoSyslog compares the whole header:

```
auth-token("Bearer s3cr3t")
```

Since the token is sent in clear text, use this option only over HTTPS, that is, together with `transport(tls)` .

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

host-override()

Type:	string
Default:	

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

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

ip() or localip()

Type:	string
Default:	`0.0.0.0`

Description: The IP address to bind to. By default, AxoSyslog listens on every available interface. Note that this is not the address where messages are accepted from.

ip-freebind()

Type:	yes or no
Default:	no

Description: Linux-only. Enables the `IP_FREEBIND` socket option, allowing AxoSyslog to bind to a non-local IP address. Useful when the address isn't yet configured on any interface (for example, during failover scenarios) so the source can listen before the address becomes available. Has no effect on platforms that don't support `IP_FREEBIND`.

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 AxoSyslog is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the AxoSyslog 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)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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:	The value of `max-connections()` multiplied by `min-iw-size-per-reader()`

Description: The size of the source window, that is, the maximum number of in-flight messages the source permits before flow control is enforced. This applies only when `flags(flow-control)` is set. For details, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-connections()

Type:	number
Default:	100

Description: The maximum number of simultaneous client connections that the source accepts.

max-request-size()

Type:	number [bytes]
Default:	The value of `log-msg-size()` multiplied by 1000

Description: The maximum size of a single HTTP request, in bytes. AxoSyslog rejects larger requests with `413 Payload Too Large` and logs an error.

For compressed requests, this limit applies to the decompressed body as well.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

port() or localport()

Type:	number
Default:	80 for `transport(tcp)`, 443 for `transport(tls)`

Description: The port number to bind to. Make sure to enable this port on the firewall of the AxoSyslog host.

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	`yes` or `no`
Default:	yes

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

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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.

Warning

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

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

so-sndbuf()

Type:	number
Default:	0

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

tags()

Type:	string
Default:	

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

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

Warning

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 AxoSyslog 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.

Warning

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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog version 3.4 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.

tls()

Type:	TLS options
Default:	

Description: The TLS settings of the source, for example, the certificate and the private key that AxoSyslog shows to the clients. Setting `transport(tls)` requires a `tls()` block. For details on the individual options, see TLS options.

```
tls(
    key-file("/etc/syslog-ng/server.key")
    cert-file("/etc/syslog-ng/server.crt")
)
```

The following TLS options are available: `ca-dir()`, `cert-file()`, `cipher-suite()`, `crl-dir()`, `dhparam-file()`, `ecdh-curve-list()`, `key-file()`, `peer-verify()`, `pkcs12-file()`, `ssl-options()`, `trusted-dn()`, and `trusted-keys()`.

> **Note**
>
> **This is a narrower set than what the `network()` and `syslog()` sources accept. In particular, `ca-file()` is not available, use `ca-dir()` instead. The `trusted-fingerprints()`, `ocsp-stapling-verify()`, `sigalgs()`, and `keylog-file()` options are not available either.**

transport()

Type:	`tcp` or `tls`
Default:	`tcp`

Description: Sets whether the source accepts plain HTTP (`tcp`) or HTTPS (`tls`) connections. Setting `transport(tls)` requires a `tls()` block, and changes the default port to 443.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

version()

Type:	string
Default:	`8.15.0`

Description: The Elasticsearch version that AxoSyslog reports to the clients in the version handshake. The Elastic clients refuse a server whose major and minor version is older than their own, unless you enable the `allow_older_versions` option on the client side. Set this option to the version your clients expect.

7.8 file: Collect 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 AxoSyslog 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 AxoSyslog is restarted, it records the position of the last sent log message in the `/opt/syslog-ng/var/syslog-ng.persist/var/lib/syslog-ng/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, AxoSyslog 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.**

7.8.1 Notes on reading kernel messages

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

- The kernel usually sends log messages to a special file (`/dev/kmsg` on BSDs, `/proc/kmsg` on Linux). The `file()` driver reads log messages from such files. The AxoSyslog 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 AxoSyslog 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 AxoSyslog in parallel. Also note that running two processes reading `/proc/kmsg` at the same time might result in dead-locks.
- When using AxoSyslog to read messages from the `/proc/kmsg` file, AxoSyslog 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));
```

7.8.2 file() source options

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

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no

Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-level()

Type:	string
Default:	notice

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

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`

- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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, AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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()`.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the

`syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

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, AxoSyslog 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()`, AxoSyslog 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`.

> **Warning**
>
> **If the `multi-line-garbage()` option is set, AxoSyslog discards lines between the line matching the `multi-line-garbage()` and the next line matching `multi-line-prefix()`.**

multi-line-mode()

Type:	`indented`, `prefix-garbage`, `prefix-suffix`, `regexp`, `smart`
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The AxoSyslog application provides the following methods to process multi-line messages:

- `indented`: 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.

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

- `prefix-garbage` : 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.

- `prefix-suffix` : 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.

- `smart` : The `smart` mode recognizes multi-line data backtraces even if they span multiple lines in the input. The backtraces are converted to a single log message for easier analysis. Backtraces for the following programming languages are recognized : Python, Java, JavaScript, PHP, Go, Ruby, and Dart.

  `smart` mode is available in AxoSyslog version 4.2 and newer.

  The regular expressions to recognize these programming languages are specified in an external file called `/usr/share/syslog-ng/smart-multi-line.fsm` (installation path depends on configure arguments), in a format that is described in that file.

Note

To format multi-line messages to your individual needs, consider the following:

- 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, AxoSyslog ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the `multi-line-garbage()` option.

Note

To format multi-line messages to your individual needs, consider the following:

- 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:2
5)
           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:2
5)
           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 AxoSyslog trying to interpret the date in the message.

multi-line-suffix()

Type:	regular expression
Default:	empty string

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

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

multi-line-timeout()

Type:	nonnegative integer (seconds)
Default:	0 (off)

Description: When using a multi-line mode (such as `regexp` or `indented`), sets the maximum time AxoSyslog waits for the next line of a multi-line message before flushing the partial message it has buffered. `0` disables the timeout. Set this option to avoid stuck partial messages when a producer pauses between lines.

normalize-hostnames()

Accepted values:	`yes` , `no`

Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()` , AxoSyslog will not read anymore from this pipe and displays the following error message:

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

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	`yes` or `no`
Default:	yes

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

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs.

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.9 Hypr Audit Trail and Hypr App Audit Trail

AxoSyslog can fetch events from the Hypr REST API using the `hypr-audit-trail()` and `hypr-app-audit-trail()` source drivers.

- `hypr-audit-trail()` : is a source driver that pulls messages from the Hypr API, associated to any RP Application ID.
- `hypr-app-audit-trail()` : is a source driver that pulls messages from the Hypr API, but only those associated to a specific RP Application ID.

Prerequisites

- AxoSyslog version 4.2.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

- This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Hypr Audit Trail

The `hypr-audit-trail()` source queries the Hypr API for the list of potential applications at startup, then monitors the audit trail for each of the detected applications.

> Note: Applications that are registered after `syslog-ng` is started are not recognized.

To start following those audit trails, you must restart `syslog-ng` .

Example minimal configuration:

```
source s_hypr {
    hypr-audit-trail(
        url('https://<custom domain>.hypr.com')
        bearer-token('<base64 encoded bearer token>')
    );
};
```

A more detailed example:

```
source s_hypr {
    hypr-audit-trail(
        url('https://<custom domain>.hypr.com')
        bearer-token('<base64 encoded bearer token>')
        page-size(<number of results to return in a single page>)
        initial-hours(<number of hours to search backward on initial fetch>)
        application-skip-list('HYPRDefaultApplication',
'HYPRDefaultWorkstationApplication')
        log-level('INFO')
        flags(<optional flags passed to the source>)
        ignore-persistence(<yes/no>)
    );
};
```

Available options:

- `url()` : custom URL for Hypr API access (`'https://<custom domain>.hypr.com'`)
- `bearer-token()` : base64 encoded authentication token from Hypr
- `page-size()` : number of results to return in a single page (optional - defaults to `100`)
- `initial-hours()` : number of hours to search backward on initial fetch (optional - defaults to `4`)
- `application-skip-list()` : list of rpAppIds not to retrieve from Hypr (optional - defaults to `'HYPRDefaultApplication', 'HYPRDefaultWorkstationApplication'`)
- `log-level()` : logging level, possible values: `"DEBUG"` , `"INFO"` , `"WARNING"` , `"ERROR"` , `"CRITICAL"` (optional - defaults to `"INFO"`)
- `flags()` : flags passed to the source, can be used for example to disable message parsing with `flags(no-parse)` (optional - defaults to empty)

- `ignore-persistence()` : ignores the saved value in the persist file, and starts querying from the current time (optional - defaults to no)

Hypr App Audit Trail

The `hypr-app-audit-trail()` monitors the audit trail for one specific RP Application ID. This driver requires the `rp-app-id()` parameter in order to operate.

Available options:

- `url()` : custom URL for Hypr API access (`'https://<custom domain>.hypr.com'`)
- `bearer-token()` : base64 encoded authentication token from Hypr
- `rp-app-id()` : the RP Application ID for the application to monitor
- `page-size()` : number of results to return in a single page (optional - defaults to `100`)
- `initial-hours()` : number of hours to search backward on initial fetch (optional - defaults to `4`)
- `log-level()` : logging level, possible values: `"DEBUG"` , `"INFO"` , `"WARNING"` , `"ERROR"` , `"CRITICAL"` (optional - defaults to `"INFO"`)
- `flags()` : flags passed to the source, can be used for example to disable message parsing with `flags(no-parse)` (optional - defaults to empty)
- `ignore-persistence()` : ignores the saved value in the persist file, and starts querying from the current time (optional - defaults to `no`)

Acknowledgements

This documentation page is based on the README file of the `hypr-audit-trail()` source, written by Dan Elder.

7.10 internal: Collect internal messages

All messages generated internally by AxoSyslog use the `internal()` source. To collect warnings, errors and notices from AxoSyslog itself, include this source in one of your source statements. The AxoSyslog application issues a warning upon startup if none of the defined log paths reference this driver.

```
internal()
```

The AxoSyslog 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)

Example: Using the internal() driver

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

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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 version 3.1 and later.

use-fqdn()

Type:	yes or no
Default:	no

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

use-syslogng-pid()

Type:	`yes`, `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.11 Jellyfin logs

You can use the `jellyfin()` source to read Jellyfin logs from its log file output.

Prerequisites

- AxoSyslog version 4.7.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `jellyfin()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/jellyfin/jellyfin.conf on GitHub.

Configuration

Example minimal configuration:

```
source s_jellyfin {
    jellyfin(
    base-dir("/path/to/my/jellyfin/root/log/dir")
    filename-pattern("log_*.log")
    );
};
```

For more details about Jellyfin logging, see:

- https://jellyfin.org/docs/general/administration/configuration/#main-configuration
- https://jellyfin.org/docs/general/administration/configuration/#log-directory

As the `jellyfin()` source is based on the `wildcard-file()` source , you can use the `wildcard-file()` source options .

7.12 kubernetes: Collect and parse the Kubernetes CRI (Container Runtime Interface) format

The `kubernetes()` source collects container logs managed by the Kubelet. It reads plain-text and JSON-formatted container logs (as described in the Container Runtime Interface (CRI) design proposal), for example, from the `/var/log/containers` or `/var/log/pods` files, and enriches them with various metadata retrieved from the Kubernetes API.

By default, it reads the `/var/log/containers` folder and extracts:

- the log content, and
- Kubernetes metadata, for example, namespace, pod, and container information.

The Kubernetes-related metadata is available in name-value pairs with the `.k8s.` prefix. The following table shows the retrieved metadata and their source:

`syslog-ng` name-value pair	source
`.k8s.namespace_name`	Container log file name.
`.k8s.pod_name`	Container log file name.
`.k8s.pod_uuid`	Container log file name or python kubernetes.client.CoreV1Api.
`.k8s.container_name`	Container log file name or python kubernetes.client.CoreV1Api.
`.k8s.container_id`	Container log file name.
`.k8s.container_image`	python kubernetes.client.CoreV1Api.
`.k8s.container_hash`	python kubernetes.client.CoreV1Api.
`.k8s.docker_id`	python kubernetes.client.CoreV1Api.
`.k8s.labels.*`	python kubernetes.client.CoreV1Api.
`.k8s.annotations.*`	python kubernetes.client.CoreV1Api.

Prerequisites

- AxoSyslog version 3.37 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"`, which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

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

- This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu, or the `axosyslog-python` package on RHEL and compatible distributions. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

Declaration

```
kubernetes(
     base-dir("<pathname>")
 );
```

kubernetes() source options

The `kubernetes()` source has the following options:

base-dir()

Type:	path without filename
Default:	`/var/log/containers`

Description: The path to the directory that contains the log files, for example, `base-dir("/var/log/pods")`.

cluster-name()

Type:	string

Default:	`k8s`

Description: The name of the Kubernetes cluster.

key-delimiter()

Type:	character
Default:	`.`

Description: The delimiter character to use when parsing flattened keys. Supports Only single characters.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

The `prefix()` option is optional and its default value is `".k8s."`.

7.13 linux-audit: Collect 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 AxoSyslog.**

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

Declaration

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

Example: Using the linux-audit() driver

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

linux-audit() source options

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

filename()

Type:	path
Default:	

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

7.14 mbox: Convert local email messages to log messages

Using the `mbox()` driver, AxoSyslog can read email messages from local mbox files, and convert them to multiline log messages.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `mbox()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/mbox/mbox.conf on GitHub.

Configuration

This driver has only one required option, the filename of the mbox file. The following example reads the emails of the root user on the AxoSyslog host.

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

mbox() source options

The `mbox()` 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

7.15 mqtt: receiving messages from an MQTT broker

You can use the `mqtt()` source to fetch messages from MQTT brokers.

The `mqtt()` source builds on the MQTT protocol , and uses its client and broker entities.

> **Note**
>
> **The rest of this chapter and its sections build on your familiarity with the MQTT protocol, the concept of client and broker entities, and how these entities function within an MQTT system.**

Prerequisites

- AxoSyslog version 3.35.0 or later.
- This feature requires a separate module. Install the `axosyslog-mod-mqtt` package on Debian/Ubuntu , or the `axosyslog-mqtt` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Using the current implementation of the `mqtt()` source has the following prerequisites:

- Installing the `eclipse-paho-mqtt-c` library.

  > **Note**
  >
  > **The default package manager for some Linux operating systems contains the `eclipse-paho-mqtt-c` library, but depending on your OS, you may have to install the library manually. For more information about how you can download and install the `eclipse-paho-mqtt-c` library, see Eclipse Paho on the Eclipse Foundation website.**

- Having a `broker` entity in a functional MQTT system.

  > **Note**
  >
  > **In your configuration, you will specify the `broker` entity of your MQTT system in the `address()` option of your `mqtt()` source.**

Limitations

Using the `mqtt()` source of AxoSyslog has the following limitations:

- You cannot use the `mqtt()` source without installing the `eclipse-paho-mqtt-c` library.

  For more information about how you can download and install the `eclipse-paho-mqtt-c` library, see Eclipse Paho on the Eclipse Foundation website.

- The current implementation of the `mqtt()` source supports versions 3.1 and 3.1.1 of the MQTT protocol.

Declaration

```
source s_mqtt{
    mqtt(
        address("tcp://<hostname>:<port-number>")
        topic("<topic-name>")
    );
};
```

Starting with AxoSyslog version 4.7, `mqtt()` source automatically sets the `${MQTT_TOPIC}` name-value pair for the messages it receives. This is useful when the name of the topic contains MQTT wildcards (`$` , `+` , `#`). For example:

```
log {
    source { mqtt(topic("#")); };
    destination { stdout(template("${MQTT_TOPIC} - ${MESSAGE}\n")); };
};
```

For the list of available optional parameters, see Options of the mqtt() source.

Example: Using the mqtt() source in your configuration

The following example illustrates an `mqtt()` source configured to fetch messages from the MQTT broker running on `localhost:4444` using the `test/test topic`, and send them to the `localhost:4445` address.

```
@version: current
@include "scl.conf"
source s_mqtt {
    mqtt(
        address("tcp://localhost:4444")
        topic("test/test")
    );
};
destination d_network {
    network(
        "localhost"
        port(4445)
    );
};
log {
    source(s_mqtt);
    destination(d_network);;
};
```

7.15.1 Options of the mqtt() source

The `mqtt()` source has the following options.

Required options: `address()` and `topic()` .

address()

Type:	string
Default:	`tcp://localhost:1883`
Required:	yes

Description: Specifies the hostname or IP address, and the port number of the MQTT broker to which AxoSyslog will send the log messages.

Syntax: `<protocol type>://<host>:<port>`

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

cleansession()

Type:	`yes` or `no`
Default:	`no`

Description: This option instruments the MQTT broker to clean the session data when connecting. The session data contains information about which message was processed.

client-id()

Type:	string
Default:	`syslog-ng-source-{topic option}`
Required:	no

Description: The `client-id()` is used to identify the client to the MQTT server, which stores session data for each client. The session data can contains information regarding which message has been sent, received. It is not possible to set the `client-id()` to an empty string. To always start a new session see the `cleansession()` option.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

fetch-no-data-delay()

Type:	nonnegative float (seconds)
Default:	value of `time-reopen()`

Description: For threaded-fetcher sources, sets how long AxoSyslog waits before the next fetch attempt when the previous attempt returned no data. Smaller values reduce latency but increase CPU usage. Larger values reduce load at the cost of higher latency.

flags()

Type:	`assume-utf8` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`

Default: empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.

- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

http-proxy()

Type:	URL
Default:	N/A

Description: Specifies HTTP/HTTPS proxy for WebSocket connections.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-alive()

Type:	positive integer number (in seconds)
Default:	`60`

Description: Specifies the number of seconds that AxoSyslog keeps the connection between the broker and clients open in case there is no message traffic. When `keep-alive()` number of seconds pass, the connection is terminated, and you have to reconnect.

On the MQTT side, the keep alive function provides a workaround method to access connections that are still open.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

log-iw-size()

Type:	number
Default:	100

Description: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

password()

Type:	string
Default:	N/A

Description: The password used to authenticate on the MQTT broker.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

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`.

qos()

Type:	number
Default:	`0`
Possible values:	`0` - at most once (the fastest option) `1` - at least once (a much slower option than `0`) `2` - exactly once (the slowest option)

Description: The Quality of Service (QoS) level in MQTT messaging is an agreement between sender and receiver on the guarantee of delivering a message.

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 version 3.1 and later.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.

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.

The following options are relevant for the `mqtt()` `tls()` block: `ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, `ssl-version()`, `use-system-cert-store()`.

topic()

Type:	string
Default:	N/A

Description: Required option. Specifies the MQTT topic to subscribe to.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

username()

Type:	string
Default:	N/A

Description: The username used to authenticate on the MQTT broker.

7.16 network: Collect 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 AxoSyslog application supports TLS (Transport Layer Security, also known as SSL) over TCP. For details, see Encrypting log messages with TLS.

Declaration:

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

By default, the `network()` driver binds to `0.0.0.0`, meaning that it listens on every available IPV4 interface on the TCP/514 port. To limit accepted connections to only one interface, use the `localip()` parameter. To listen on IPv6 addresses, use the `ip-protocol(6)` option.

Example: Using the network() driver

Using only the default settings: listen on every available IPV4 interface on the TCP/514 port.

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

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

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

TCP source listening on the IPv6 localhost, port 2222:

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

A TCP source listening on a TLS-encrypted channel.

```
source s_network {
     network(
         transport("tls")
         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: Collect messages using the IETF-syslog protocol.

```
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.

See also

- Configuring AxoSyslog as a logserver
- Configuring TLS server authentication on the AxoSyslog server
- Configuring mutual TLS authentication on the AxoSyslog server

7.16.1 network() source options

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

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

dynamic-window-realloc-ticks()

Type:	nonnegative number
Default:	10

Description: When `dynamic-window-size()` is enabled, controls how many statistics ticks must pass before AxoSyslog reallocates the dynamic window across active connections. Larger values dampen reallocation churn under bursty traffic; smaller values let the window adapt more aggressively to per-connection load. For background, see Managing incoming and outgoing messages with flow-control.

dynamic-window-size()

Type:	number
Default:	0

Description: The size of the dynamic control window used during flow-control. For details on flow-control, see Dynamic flow-control.

dynamic-window-stats-freq()

Type:	nonnegative float (seconds)
Default:	0.0 (off)

Description: When `dynamic-window-size()` is set to a non-zero value, sets how often AxoSyslog samples per-connection throughput statistics that drive window reallocation. `0` turns off sampling. The interval is stored internally in milliseconds, so fractional seconds are honored.

ebpf()

Available in AxoSyslog version 4.2 and newer.

By default, the kernel chooses the receive socket for a specific UDP randomly based on the source IP/port of the sender. You can customize this algorithm using the Extended Berkeley Packet Filter (eBPF) plugin. The `ebpf()` option changes the kernel's SO_REUSEPORT algorithm so that all messages are randomly placed into one of the UDP sockets. The decision which UDP socket buffer a datagram is placed is made for every datagram, and not once for every stream. This means that messages are perfectly load-balanced across your set of UDP sockets. While this resolves the imbalance between the sockets and results in perfect load balancing, you will lose ordering between messages from the same sender, which is the price to pay for increased throughput.

```
source s_network {
    network(
        transport("udp")
        so-reuseport(1) persist-name("udp1")
        ebpf(reuseport(sockets(4)))
    );
    network(transport("udp") so-reuseport(1) persist-name("udp2"));
    network(transport("udp") so-reuseport(1) persist-name("udp3"));
    network(transport("udp") so-reuseport(1) persist-name("udp4"));
};
```

For a detailed introduction, see the Scaling syslog to 1M EPS with eBPF blog post.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)

- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

- *threaded*: The `threaded` flag enables multithreading for the source. For details on multithreading, see Multithreading and scaling.

> **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.

> **Note**
>
> **The AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

interface()

Type:	string
Default:	None

Description: Bind to the specified interface instead of an IP address. Available in 3.19 and later.

ip() or localip()

Type:	string
Default:	0.0.0.0

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

ip-freebind()

Type:	yes or no
Default:	no

Description: Linux-only. Enables the `IP_FREEBIND` socket option, allowing AxoSyslog to bind to a non-local IP address. Useful when the address isn't yet configured on any interface (for example, during failover scenarios) so the source can listen before the address becomes available. Has no effect on platforms that don't support `IP_FREEBIND`.

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 AxoSyslog is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the AxoSyslog 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)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

Note

If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

Warning

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

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)
         );
 };
```

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-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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections (except for UDP-based connection), 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 AxoSyslog clients, make sure that the window size is larger than the `flush-lines()` option set in the destination of your clients.

Example: Initial window size of a connection

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

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-connections()

Type:	number

Default:	10

Description: Specifies the maximum number of simultaneous connections. Doesn't affect UDP-based traffic, which is always treated as a single connection.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()` , AxoSyslog will not read anymore from this pipe and displays the following error message:

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

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

port() or localport()

Type:	number
Default:	In case of TCP transport: 514 In case of UDP transport: 514

Description: The port number to bind to.

program-override()

Type:	string
Default:	

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

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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.

> **Warning**
>
> When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the AxoSyslog server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In this cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.
>
> As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.

so-reuseport()

Type:	yes or no
Default:	no

Description: Enables SO_REUSEPORT on systems that support it. When enabled, the kernel allows multiple UDP sockets to be bound to the same port, and the kernel load-balances incoming UDP datagrams to the sockets. The sockets are distributed based on the hash of (srcip, dstip, srcport, dstport), so the same listener should be receiving packets from the same endpoint. For example:

```
   source {
            network(transport("udp") so-reuseport(1) port(2000) persist-
name("udp1"));
            network(so-reuseport(1) port(2000) persist-name("udp2"));
            network(so-reuseport(1) port(2000) persist-name("udp3"));
            network(so-reuseport(1) port(2000) persist-name("udp4"));
    };
```

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-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 version 3.1 and later.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

> **Warning**
>
> 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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog version 3.4 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:	`auto`, `nul-terminated`, `proxied-tcp`, `proxied-tls`, `proxied-tls-passthrough`, `proxied-udp`, `tcp`, `text-with-nuls`, `tls`, `udp`
Default:	`tcp`

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

- `auto`: Available in version 4.10 and later. Automatically detects RFC6587-style octet-count based framing in TCP traffic. In future versions this functionality will be extended to cover and detect all TCP-based protocols.
- `nul-terminated`: Available in version 4.21 and later. Uses NUL characters to separate log records instead of newline characters. For example, the following message body will be interpreted as a multi-line message: `line1\nline2\nline3\x00`

- `proxied-tcp` , `proxied-tls` , `proxied-tls-passthrough` , `proxied-udp` : Refers to the HAProxy Proxy Protocol. `proxied-udp` is available in version 4.25 and later. For details, see Proxy Protocol support.

- `text-with-nuls` : Allows embedded `NUL` characters in the message from a TCP source, that is, AxoSyslog will not delimiter the incoming messages on `NUL` characters, only on `newline` characters (contrary to `tcp` transport, which splits the incoming log on `newline` characters and `NUL` characters).

> **Note**
>
> **The AxoSyslog application does not support embedded `NUL` characters everywhere, so it is recommended that you also use `flags(no-multi-line)` that causes `NUL` characters to be replaced by space.**

> **Warning**
>
> When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the AxoSyslog server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In this cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.
>
> As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.

trim-large-messages()

Type:	`yes` or `no`
Default:	Use the global `trim-large-messages()` option, which defaults to `no` .

Description: Determines what AxoSyslog does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()` . Other drivers ignore the `trim-large-messages()` option.

- If set to `no` , AxoSyslog drops the incoming log message.

- If set to `yes`, AxoSyslog trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter f_trimmed {
      tags("trimmed");
  };
  ```

  If AxoSyslog trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in AxoSyslog version 3.21 and later.

Uses the value of the global option if not specified.

tls()

Type:	tls options
Default:	n/a

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

use-dns()

Type:	`yes`, `no`, `persist_only`
Default:	`yes`

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option. The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network

endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.16.2 Proxy Protocol support

If you connect load balancers to your AxoSyslog application, AxoSyslog identifies every connection that is connected to the load balancers identically by default, regardless of the source IP or the source protocol. Essentially, the load balancer masks the source IP unless you enable Proxy Protocol support for your proxy TLS `transport()` to inject information about the original connection into the forwarded TCP session.

For further details about the working mechanism behind the Proxy Protocol support on AxoSyslog and the configuration details, see the following sections:

7.16.2.1 The working mechanism behind the Proxy Protocol

This section describes how AxoSyslog supports the Proxy Protocol .

The working mechanism behind the Proxy Protocol

When using the Proxy Protocol during load balancing, AxoSyslog detects the information behind connections connected to the load balancer, then parses the injected information into the following macros:

Data	Macro in version 4.9 and earlier	Version 4.10 and later
Source IP of the proxy	`PROXIED_SRCIP`	`SOURCEIP`
Source port of the proxy	`PROXIED_SRCPORT`	`SOURCEPORT`
Destination IP of the proxy	`PROXIED_DSTIP`	`DESTIP`
Destination port of the proxy	`PROXIED_DSTPORT`	`DESTPORT`
IP version of the connection	`PROXIED_IP_VERSION`	`IP_PROTO`

> **Note**
>
> Consider the following about macros and headers:
>
> - When the proxy protocol header is `PROXY UNKNOWN` , no additional macros are added.
> - When AxoSyslog cannot parse a proxy protocol header, the connection is closed:
>
> ```
> [2020-11-20T17:33:22.189458] PROXY protocol header received;
> line='PROXYdsfj'
> [2020-11-20T17:33:22.189475] Error parsing PROXY protocol header;
> [2020-11-20T17:33:22.189517] Syslog connection closed; fd='13',
> client='AF_INET(127.0.0.1:51665)', local='AF_INET(0.0.0.0:6666)'
> [2020-11-20T17:33:22.189546] Freeing PROXY protocol source driver;
> driver='0x7fffcba5bcf0'
> [2020-11-20T17:33:22.189600] Closing log transport fd; fd='13'
> ```

> **Note**
>
> **Originally, the driver supported version 1 of the PROXY protocol (TCP4, TCP6, and TLS connections). PROXY protocol v2 support is available in AxoSyslog version 4.1 and later. UDP transport support for PROXY protocol v2 (`proxied-udp`) is available in version 4.25 and later.**

7.16.2.2 Proxy Protocol: configuration and output examples

This section provides information about enabling Proxy Protocol support in your `network()` source options, and an example configuration and output to illustrate how the Proxy Protocol method works in AxoSyslog.

For more information about the working mechanism of the Proxy Protocol, see The working mechanism behind the Proxy Protocol.

Enabling Proxy Protocol support for your network() source options

Unless you enable Proxy Protocol support for your `network()` source, AxoSyslog identifies every connection that is connected to the load balancers identically by default, regardless of the source IP or the source protocol.

To enable Proxy Protocol for your `network()` source, set the `transport()` parameter of your `network()` source to `proxied-tcp`, `proxied-tls-passthrough`, or `proxied-udp`, depending on your preference and configuration.

`proxied-tls` can be used in complex MITM (man in the middle) configurations, where the proxy header is sent encrypted within the same TLS session as the proxied messages.

When you enable Proxy Protocol support for your `network()` source, you can use the following configuration example with your AxoSyslog application.

Configuration

The following code sample illustrates how you can use the Proxy Protocol in your AxoSyslog configuration (using the `transport()` parameter set to `proxied-tls-passthrough`).

```
   @version: 3.35

    source s_tcp_pp {
      network (
        port(6666)
        transport("proxied-tls-passthrough")
        tls(
            key-file("/certs/certs/server/server.rsa")
            cert-file("/certs/certs/server/server.crt")
            ca-dir("/certs/certs/CA")
            peer-verify("required-trusted")
        )
      );
    };

    destination d_file {
      file("/var/log/proxy-proto.log" template("${SOURCEIP} ${SOURCEPORT} ${DESTIP}
${DESTPORT} ${IP_PROTO}\n"));
    };

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

With this configuration, the Proxy Protocol method will perform injecting the information of the original connection into the forwarded TCP session, based on the working mechanism described in The working mechanism behind the Proxy Protocol.

The following example illustrates how the parsed macros will appear in the output.

Example: Output for the PROXY TCP4 192.168.1.1 10.10.0.1 1111 2222 input header

With the `PROXY TCP4 192.168.1.1 10.10.0.1 1111 2222` input header, the output looks like this:

```
   192.168.1.1 1111 10.10.0.1 2222 4
```

Note that the macros that AxoSyslog adds to the message appear in the output.

Enabling Proxy Protocol v2 support over UDP

Available in AxoSyslog version 4.25 and later.

To receive UDP syslog messages from a load balancer that injects HAProxy Proxy Protocol v2 headers, set `transport("proxied-udp")` on your `network()` or `syslog()` source.

> **Note**
>
> `proxied-udp` **supports HAProxy Proxy Protocol version 2 only. Proxy Protocol v1 is not supported over UDP.**

Using the `network()` driver (BSD syslog/RFC3164 format):

```
source s_udp_pp {
    network(
        ip("0.0.0.0")
        port(514)
        transport("proxied-udp")
    );
};
```

Using the `syslog()` driver (IETF syslog/RFC5424 format):

```
source s_syslog_udp_pp {
    syslog(
        ip("0.0.0.0")
        port(514)
        transport("proxied-udp")
    );
};
```

After parsing, the original client address and port are available as `${SOURCEIP}`, `${SOURCEPORT}`, `${DESTIP}`, and `${DESTPORT}` macros, the same as for TCP-based proxied transports.

7.17 nodejs: Receive JSON messages from nodejs applications

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

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `nodejs()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/nodejs/plugin.conf on GitHub.

Configuration

The following example uses the default settings of the driver, listening for messages on port 9003 of every IP address of the AxoSyslog 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)
     )
 };
```

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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, AxoSyslog 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, AxoSyslog 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.

7.18 Collect OpenBSD kernel log messages

The `openbsd()` source driver collects kernel log messages on OpenBSD systems. The driver reads messages from the `/dev/klog` kernel log device using the `LIOCSFD` ioctl, which is the OpenBSD-native mechanism that the system's default `syslogd` use. Messages collected through this source are tagged with the `local+openbsd` transport.

On OpenBSD platforms the `system()` source automatically uses `openbsd()` to read kernel messages, so usually you don't need to configure it directly. Use `openbsd()` explicitly only if you want to bypass `system()` and route OpenBSD kernel messages through a custom log path.

Prerequisites

AxoSyslog supports this feature only on OpenBSD. The Debian/Ubuntu and RHEL packages that Axoflow publishes don't include it. To use it, build AxoSyslog from source on OpenBSD.

Example: Reading OpenBSD kernel logs

```
source s_openbsd_kernel {
    openbsd();
};

log {
    source(s_openbsd_kernel);
    destination { file("/var/log/kernel.log"); };
};
```

Options

The `openbsd()` source supports a minimal set of options. It doesn't accept message-format or socket-tuning options because the kernel log device delivers locally framed messages with no hostname or transport metadata to negotiate.

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

7.19 Receive logs, metrics, and traces from OpenTelemetry

AxoSyslog can receive logs, metrics, and traces from OpenTelemetry clients over the OpenTelemetry Protocol (OTLP/gRPC) .

Prerequisites

- AxoSyslog version 4.3.0 or later.
- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Example: Receiving OpenTelemetry data

The following example receives OpenTelemetry data and forwards it to an OpenTelemetry receiver. Note that by default, AxoSyslog doesn't parse the fields of the incoming messages into name-value pairs, but are only available for forwarding using the `opentelemetry()` destination. To parse the fields into name-value pairs, use the `opentelemetry()` parser .

```
log otel_forward_mode_alts {
  source {
    opentelemetry(
      port(4317)
      auth(alts())
    );
  };

  destination {
    opentelemetry(
      url("my-otel-server:12345")
      auth(alts())
    );
  };
};
```

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

```
  opentelemetry(
    port(4317)
    auth(alts())
  );
```

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_otlp {
    opentelemetry(
      url("your-otel-server:12346")
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

peer-verify()

Accepted values:	`optional-trusted` , `optional-untrusted` , `required-trusted` , `required-untrusted`
Default:	`required-trusted`

Description: Verification method of the peer, the four possible values is a combination of two properties of validation:

- Whether the peer is required to provide a certificate (required or optional prefix).
- Whether the certificate provided needs to be valid or not.

The following table summarizes the possible options and their results depending on the certificate of the peer.

The remote peer has:		
no certificate	**invalid certificate**	**valid certificate**

Local peer-verify() setting	optional-untrusted	TLS-encryption	TLS-encryption	TLS-encryption
	optional-trusted	TLS-encryption	rejected connection	TLS-encryption
	required-untrusted	rejected connection	TLS-encryption	TLS-encryption

Local peer-verify() setting	required-trusted	reject connection	reject connection	TLS-encryption

For untrusted certificates only the existence of the certificate is checked, but it does not have to be valid – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

channel-args()

Type:	arrow list

Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

concurrent-requests()

Type:	integer
Default:	2

Description: Configures the maximal number of in-flight gRPC requests per worker. Setting this value in the range of 10s or 100s is recommended when there are a high number of clients sending simultaneously. Ideally, `workers() * concurrent-requests()` should be greater than or equal to the number of clients, but this can increase the memory usage.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string

Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)`.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

flags()

Type:	`assume-utf8`, `check-hostname`, `check-program`, `dont-store-legacy-msghdr`, `empty-lines`, `exit-on-eof`, `expect-hostname`, `guess-timezone`, `ignore-aux-data`, `kernel`, `no-header`, `no-hostname`, `no-multi-line`, `no-parse`, `sanitize-utf8`, `store-legacy-msghdr`, `store-raw-message`, `syslog-protocol`, `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

ip()

Type:	string
Default:	`0.0.0.0`

Available in AxoSyslog 4.23 and later.

Description: The IP address to bind to. By default, AxoSyslog listens on every available interface. Note that this is not the address where messages are accepted from.

keep-alive()

Type:	`yes` or `no`
Default:	`yes`

Available in AxoSyslog 4.20 and later.

Description: Client connections can be kept alive during reload, avoiding unnecessary retry backoffs and other error messages on the client side.

keep-hostname()

The `opentelemetry()` source ignores this option and uses the address of the OTLP peer as the HOST.

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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()`.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the

> `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

`mode()`

Type:	`logmessage` or `filterx-dict`
Default:	`logmessage`

Available in AxoSyslog 4.28 and later.

Description: Determines how AxoSyslog makes the contents of incoming OpenTelemetry log records available for processing.

- `logmessage` : AxoSyslog stores the record in the `${.otel_raw.log}` , `${.otel_raw.resource}` , and `${.otel_raw.scope}` name-value pairs. To work with them in FilterX, map them to OTEL objects first, as described in Handle OpenTelemetry log records.

- `filterx-dict` : AxoSyslog converts the record directly into three declared FilterX variables called `log` , `resource` , and `scope` , each holding a plain FilterX dictionary . Use them in your FilterX block without any input mapping.

  Because this skips serializing and deserializing the `${.otel_raw.*}` name-value pairs, `mode(filterx-dict)` is significantly faster than `mode(logmessage)` when you process the records in FilterX.

Note the following points about `mode(filterx-dict)` :

- The `log`, `resource`, and `scope` variables are plain dictionaries, not the `otel_logrecord`, `otel_resource`, and `otel_scope` objects that the OTEL FilterX functions create.
- The mode affects only log records. Metrics and traces are always stored as name-value pairs.
- Messages sent by a AxoSyslog `syslog-ng-otlp()` destination are recognized and parsed the same way in both modes.
- The `${.otel_raw.*}` name-value pairs are not set, so configurations that depend on them – for example, forwarding the received data unchanged to an `opentelemetry()` destination – require `mode(logmessage)`.

normalize-hostnames()

Accepted values:	`yes`, `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

`port()`

The port number to receive incoming connections. Default value: 4317

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to process the messages of the source. Increasing the number of worker threads can drastically improve the performance of the source.

7.20 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 AxoSyslog allows you read the results of periodical osquery queries (from the `/var/log/osquery/osqueryd.results.log` file) and automatically parse the messages. 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 AxoSyslog.

The AxoSyslog application automatically adds the `.osquery.` prefix to the name of the fields the extracted from the message.

Prerequisites

- AxoSyslog version 3.10 and later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `osquery()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/osquery/plugin.conf on GitHub.
- AxoSyslog must be compiled with JSON-support enabled.

Example: Using the osquery() driver with the default settings

The following AxoSyslog 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.

```
@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:

```
@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 AxoSyslog 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":"43cd1c6a7d0c283e2
1e026a53e619b2e582e94ee","config_valid":"1","counter":"4","extensions":"active","in
stance_id":"d0c3eb0d-f8e0-4bea-868b-
18a2c61b438d","pid":"19764","resident_size":"26416000","start_time":"1500629552","s
ystem_time":"223","user_time":"476","uuid":"4C4C4544-004D-3610-8043-
C2C04F4D3332","version":"2.5.0","watcher":"19762"},"action":"added"}
```

AxoSyslog configuration

```
@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","resi
dent_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'
```

7.20.1 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

Default value:

`.osquery.` option.

7.21 pacct: Collect process accounting logs on Linux

AxoSyslog 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).

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`), AxoSyslog 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.

Prerequisites

- AxoSyslog version 3.2.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `pacct()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/pacct/plugin.conf on GitHub.
- The AxoSyslog 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, AxoSyslog automatically loads the available modules.
- Process accounting must be running on the host. You can enable it with the `accton` command.

Configuration

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

```
@version: 4.29
@include "scl.conf"
source s_pacct { pacct(); };
...
log { source(s_pacct); destination(...); };
```

pacct() source 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 – AxoSyslog 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, AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

7.22 Pi-hole Faster Than Light logs

AxoSyslog can collect logs of the Pi-hole FTL (Faster Than Light) application.

Prerequisites

- AxoSyslog version 4.6.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `pihole-ftl()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/pihole/pihole.conf on GitHub.

Configuration

```
source s_pihole_ftl {
  pihole-ftl();
};
```

By default, the source reads the `/var/log/pihole/FTL.log` file. If the root directory of your Pi-hole installation is different, specify the directory where the FTL.log file is with the `dir()` option. You can find the root log directory by selecting **Tools > Pi-hole diagnosis** in the Pi-hole application. Otherwise, the `pihole-ftl()` source has the same parameters as the `file()` source .

7.23 pipe: Collect 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 version 3.0.2, pipes are created automatically. In earlier versions, you had to create the pipe using the `mkfifo(1)` command.**

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

> **Warning**
>
> - It is not recommended to use `pipe()` on anything else than real pipes.
> - By default, AxoSyslog uses the `flags(no-hostname)` option for pipes, meaning that AxoSyslog 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)`.

Example: Using the pipe() driver

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

7.23.1 pipe() source options

The `pipe` driver has the following options:

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no

Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

create-dirs()

Type:	yes or no
Default:	no

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

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp,

host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.

- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

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, AxoSyslog 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()`, AxoSyslog 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`.

> **Warning**
>
> **If the `multi-line-garbage()` option is set, AxoSyslog discards lines between the line matching the `multi-line-garbage()` and the next line matching `multi-line-prefix()`.**

multi-line-mode()

Type:	`indented`, `prefix-garbage`, `prefix-suffix`, `regexp`, `smart`
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The AxoSyslog application provides the following methods to process multi-line messages:

- `indented`: 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.

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

- `prefix-garbage`: 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.

- `prefix-suffix`: 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.

- `smart`: The `smart` mode recognizes multi-line data backtraces even if they span multiple lines in the input. The backtraces are converted to a single log message for easier analysis. Backtraces for the following programming languages are recognized : Python, Java, JavaScript, PHP, Go, Ruby, and Dart.

  `smart` mode is available in AxoSyslog version 4.2 and newer.

  The regular expressions to recognize these programming languages are specified in an external file called `/usr/share/syslog-ng/smart-multi-line.fsm` (installation path depends on configure arguments), in a format that is described in that file.

Note

To format multi-line messages to your individual needs, consider the following:

- 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, AxoSyslog ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the `multi-line-garbage()` option.

Note

To format multi-line messages to your individual needs, consider the following:

- 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:2
5)
           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:2
5)
           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 AxoSyslog 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.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

optional()

Type:	yes or no
Default:	

Description: Instruct AxoSyslog to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()`, AxoSyslog will not read anymore from this pipe and displays the following error message:

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

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

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` .

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.` , using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

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 version 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.

trim-large-messages()

Type:	`yes` or `no`
Default:	Use the global `trim-large-messages()` option, which defaults to `no`.

Description: Determines what AxoSyslog does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, AxoSyslog drops the incoming log message.
- If set to `yes`, AxoSyslog trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

```
filter f_trimmed {
    tags("trimmed");
};
```

If AxoSyslog trims a log message, it sends a debug-level log message to its `internal()` source.

As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in AxoSyslog version 3.21 and later.

Uses the value of the global option if not specified.

use-dns()

Type:	`yes` , `no` , `persist_only`
Default:	`yes`

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.24 program: Receive 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);
```

For the list of available optional parameters, see program() source options.

Example: Using the program() driver

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

> **Note**
>
> **The program is restarted automatically if it exits.**

7.24.1 program() source options

The `program` driver has the following options:

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no

Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)`.

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`

- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

inherit-environment()

Type:	`yes` or `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, AxoSyslog). Use `inherit-environment(no)` to prevent this.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

normalize-hostnames()

Accepted values:	`yes` , `no`

Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

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`.

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs.

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 version 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.

trim-large-messages()

Type:	`yes` or `no`
Default:	Use the global `trim-large-messages()` option, which defaults to `no`.

Description: Determines what AxoSyslog does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, AxoSyslog drops the incoming log message.
- If set to `yes`, AxoSyslog trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter f_trimmed {
      tags("trimmed");
  };
  ```

  If AxoSyslog trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in AxoSyslog version 3.21 and later.

Uses the value of the global option if not specified.

use-dns()

Type:	`yes` , `no` , `persist_only`
Default:	`yes`

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.25 python: writing server-style Python sources

The Python source allows you to write your own source in Python. You can import external Python modules to receive or fetch the messages. Since many services have a Python library, the Python source makes integrating AxoSyslog very easy and quick.

You can write two different type of sources in Python:

- Server-style sources that receives messages. Write server-style sources if you want to use an event-loop based, nonblocking server framework in Python, or if you want to implement a custom loop.
- Fetcher-style sources that actively fetch messages. In general, write fetcher-style sources (for example, when using simple blocking APIs), unless you explicitly need a server-style source.

This section describes server-style sources. For details on fetcher-style sources, see python-fetcher: writing fetcher-style Python sources.

The following points apply to using Python blocks in AxoSyslog in general:

- Python parsers and template functions are available in AxoSyslog version 3.10 and later.

  Python destinations and sources are available in AxoSyslog version 3.18 and later.
- Supported Python versions: 2.7 and 3.4+ (if you are using pre-built binaries, check the dependencies of the package to find out which Python version it was compiled with).
- The Python block must be a top-level block in the AxoSyslog configuration file.
- If you store the Python code in a separate Python file and only include it in the AxoSyslog configuration file, make sure that the PYTHONPATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start AxoSyslog manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`.

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

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

- The Python block can contain multiple Python functions.
- Using Python code in AxoSyslog can significantly decrease the performance of AxoSyslog, especially if the Python code is slow. In general, the features of AxoSyslog are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The AxoSyslog application does not do any of this.
- Python error messages are available in the `internal()` source of AxoSyslog.
- You can access the name-value pairs of AxoSyslog directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of AxoSyslog. For details, see Logging from your Python code.

Note

Starting with 3.26, AxoSyslog assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your AxoSyslog configuration, add a unique `persist-name()` to each source or destination, otherwise AxoSyslog will not start. For example:

```
    log {
         source { python(class(PyNetworkSource) options("port" "8080") persist-
 name("<unique-string>); };
         source { python(class(PyNetworkSource) options("port" "8081")); };
     };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name` . For example:

```
    from syslogng import LogSource
       class PyNetworSource(LogSource):
         @staticmethod
         def generate_persist_name(options):
             return options["port"]
         def run(self):
             pass
         def request_exit(self):
             pass
```

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

Python sources consist of two parts. The first is a AxoSyslog source object that you define in your AxoSyslog configuration and use in the log path. This object references a Python class, which is the second part of the Python source. The Python class receives

or fetches the log messages, and can do virtually anything that you can code in Python. You can either embed the Python class into your AxoSyslog configuration file, or store it in an external Python file .

```
source <name_of_the_python_source>{
     python(
         class("<name_of_the_python_class_executed_by_the_source>")
         options(
             "option1" "value1",
             "option2" "value2"
         )
     );
 };

 python {
 from syslogng import LogSource
 from syslogng import LogMessage

 class <name_of_the_python_class_executed_by_the_source>(LogSource):
     def init(self, options): # optional
         print("init")
         print(options)
         self.exit = False
         return True

     def deinit(self): # optional
         print("deinit")

     def run(self): # mandatory
         print("run")
         while not self.exit:
             # Must create a message
             msg = LogMessage("this is a log message")
             self.post_message(msg)

     def request_exit(self): # mandatory
         print("exit")
         self.exit = True
 };
```

Methods of the python() source

Server-style Python sources must be inherited from the `syslogng.LogSource` class, and must implement at least the `run` and `request_exit` methods. Multiple inheritance is allowed, but only for pure Python super classes.

You can implement your own event loop, or integrate the event loop of an external framework or library, for example, KafkaConsumer , Flask , Twisted engine , and so on.

To post messages, call `LogSource::post_message()` method in the `run` method.

For the list of available optional parameters, see python() and python-fetcher() source options.

init(self, options) method (optional)

The AxoSyslog application initializes Python objects every time when it is started or reloaded. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your source to work.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

When this method returns with False, AxoSyslog does not start. It can be used to check options and return False when they prevent the successful start of the source.

`options` : This optional argument contains the contents of the `options()` parameter of the AxoSyslog configuration object as a Python dictionary.

run(self) method (mandatory)

Use the `run` method to implement an event loop, or start a server framework or library. Create `LogMessage` instances in this method, and pass them to the log paths by calling `LogSource::post_message()` .

Currently, `run` stops permanently if an unhandled exception happens.

For details on parsing and posting messages, see Python LogMessage API.

request_exit(self) method (mandatory)

The AxoSyslog application calls this method when AxoSyslog is shut down or restarted. The `request_exit` method must shut down the event loop or framework, so the `run` method can return gracefully. If you use blocking operations within the `run()` method, use `request_exit()` to interrupt those operations and set an exit flag, otherwise AxoSyslog is not able to stop. Note that AxoSyslog calls the `request_exit` method from a thread different from the source thread.

close_batch(self)

Closes the current source-side batch. Source-side batching helps AxoSyslog to effectively process a larger chunk of messages, instead of processing messages each message. For example, when feeding a destination queue and instead of taking a lock on the queue for every message (causing contention), we only take it once per batch.

The native drivers built into AxoSyslog typically close batches once every mainloop iteration, allowing a single iteration to process multiple messages. For instance, when receiving multiple messages in a single TCP datagram, all of those messages can be processed as a part of the same batch.

In Python-based log sources, a batch will automatically be closed after every message posted via `post_message()`, except if `self.auto_close_batches` is set to `False` during initialization. In case `self.auto_close_batches` is set to `False`, the driver has to call `close_batch()` explicitly, preferably at a natural boundary between incoming batches of messages. A good example is when we retrieve several messages via the same HTTP REST call, then the right time to close the batch would be after the last message in the response is posted.

The deinit(self) method (optional)

This method is executed when AxoSyslog is stopped or reloaded. This method does not return a value.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

set_transport_name(self, name)

Set the transport name used to retrieve messages This function can be called to customize the `${TRANSPORT}` name-value pair .

7.25.1 Python LogMessage API

The LogMessage API allows you to create LogMessage objects in Python sources, parse syslog messages, and set the various fields of the log message.

LogMessage() method: Create log message objects

You can use the `LogMessage()` method to create a structured log message instance. For example:

```
   from syslogng import LogMessage

    msg = LogMessage() # Initialize an empty message with default values (recvd
timestamp, rcptid, hostid, ...)
    msg = LogMessage("string or bytes-like object") # Initialize a message and set
its ${MESSAGE} field to the specified argument
```

You can also explicitly set the different values of the log message. For example:

```
   msg["MESSAGE"] = "message"
    msg["HOST"] = "hostname"
```

You can set certain special field (timestamp, priority) by using specific methods.

Note the following points when creating a log message:

- When setting the hostname, AxoSyslog takes the following hostname-related options of the configuration into account: `chain-hostnames()`, `keep-hostname()`, `use-dns()`, and `use-fqdn()`.
- Python sources ignore the `log-msg-size()` option.
- The AxoSyslog application accepts only one message from every `LogSource::post_message()` or `fetch()` call, batching is currently not supported. If your Python code accepts batches of messages, you must pass them to AxoSyslog one-by-one. Similarly, if you need to split messages in the source, you must do so in your Python code, and pass the messages separately.

- Do not reuse or store LogMessage objects after posting (calling `post_message()`) or returning the message from `fetch()`.

parse() method: Parse syslog messages

The `parse()` method allows you to parse incoming messages as syslog messages. By default, the `parse()` method attempts to parse the message as an IETF-syslog (RFC5424) log message. If that fails, it parses the log message as a BSD-syslog (RFC3164) log message. Note that AxoSyslog takes the parsing-related options of the configuration into account: `flags()`, `keep-hostname()`, `recv-time-zone()`.

If `keep-hostname()` is set to `no`, AxoSyslog ignores the hostname set in the message, and uses the IP address of the AxoSyslog host as the hostname (to use the hostname instead of the IP address, set the `use-dns()` or `use-fqdn()` options in the Python source).

```
   msg_ietf = LogMessage.parse('<165>1 2003-10-11T22:14:15.003Z
mymachine.example.com evntslog - ID47 [exampleSDID@32473 iut="3"
eventSource="Application" eventID="1011"] An application event log entry',
self.parse_options)
    msg_bsd = LogMessage.parse('<34>Oct 11 22:14:15 mymachine su: \'su root\'
failed for lonvick on /dev/pts/8', self.parse_options)
```

set_pri() method

You can set the priority of the message with the `set_pri()` method.

```
   msg.set_pri(165)
```

set_timestamp() method

You can use the `set_timestamp()` method to set the date and time of the log message.

```
   timestamp = datetime.fromisoformat("2018-09-11T14:49:02.100+02:00")
    msg.set_timestamp(timestamp) # datetime object, includes timezone information
```

In Python 2, timezone information cannot be attached to the datetime instance without using an external library. The AxoSyslog represents naive datetime objects in UTC.

In Python 3, naive and timezone-aware datetime objects are both supported.

7.25.2 python() and python-fetcher() source options

The `python()` and `python-fetcher()` drivers have the following options.

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

class()

Type:	string
Default:	N/A

Description: The name of the Python class that implements the source, for example:

```
python(
     class("MyPythonSource")
 );
```

If you want to store the Python code in an external Python file, the `class()` option must include the name of the Python file containing the class, without the path and the .py extension, for example:

```
python(
     class("MyPythonfilename.MyPythonSource")
 );
```

For details, see Python code in external files

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

fetch-no-data-delay()

Available on `python-fetcher()` only.

Type:	integer [seconds]
Default:	-1 (disabled)

Description: If the `fetch` method of a `python-fetcher()` source returns with the `LogFetcher.FETCH_NO_DATA` constant, the source waits `fetch-no-data-delay()` seconds before calling the `fetch` method again. If you want to call the `fetch` method sooner, set the `fetch-no-data-delay()` option to the number of seconds to wait before calling the `fetch` method.

flags()

Type:	`assume-utf8` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

For the `python()` and `python-fetcher()` sources you can also set the `check-hostname` flag, which is equivalent with the `check-hostname()` global option , but only applies to this source.

The flags and the hostname-related options (for example, `use-dns`) set in the configuration file influence the behavior of the `LogMessage.parse()` method of the Python source. They have no effect if you set the message or the hostname directly, without using `LogMessage.parse()` .

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

imports() (DEPRECATED)

Obsolete alias for `loaders()`. When set, behaves identically to `loaders()`. Use `loaders()` instead.

internal()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

log-iw-size()

Type:	number

Default:	100

Description: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

loaders()

Type:	list of python modules
Default:	empty list

Description: The AxoSyslog application imports Python modules specified in this option, before importing the code of the Python class. This option has effect only when the Python class is provided in an external Python file. This option has no effect when the

Python class is provided within the AxoSyslog configuration file (in a `python{}` block). You can use the `loaders()` option to modify the import mechanism that imports Python class. For example, that way you can use hy in your Python class.

```
python(class(usermodule.HyParser) loaders(hy))
```

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

options()

Type:	string
Default:	N/A

Description: This option allows you to pass custom values from the configuration file to the Python code. Enclose both the option names and their values in double-quotes. The Python code will receive these values during initialization as the `options` dictionary. For example, you can use this to set the IP address of the server from the configuration file, so it is not hard-coded in the Python object.

```
python(
     class("MyPythonClass")
     options(
         "host" "127.0.0.1"
         "port" "1883"
         "otheroption" "value")
 );
```

For example, you can refer to the value of the host field in the Python code as `options["host"]`. Note that the Python code receives the values as strings, so you might have to cast them to the type required, for example: `int(options["port"])`

> **Note**
>
> From version 3.27, AxoSyslog supports the arrow syntax for declaring custom Java and Python options. You can alternatively declare them using a similar syntax:
>
> ```
> options(
>   "host" => "localhost"
>   "port" => "1883"
>   "otheroption" => "value"
> )
> ```

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

Note

Starting with 3.26, AxoSyslog assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your AxoSyslog configuration, add a unique `persist-name()` to each source or destination, otherwise AxoSyslog will not start. For example:

```
   log {
        source { python(class(PyNetworkSource) options("port" "8080") persist-
name("<unique-string>); };
        source { python(class(PyNetworkSource) options("port" "8081")); };
    };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:

```
   from syslogng import LogSource
      class PyNetworSource(LogSource):
        @staticmethod
        def generate_persist_name(options):
            return options["port"]
        def run(self):
            pass
        def request_exit(self):
            pass
```

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 version 3.1 and later.

time-reopen()

Accepted values:	number [seconds]
Default:	1

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

Available in `python-fetcher()` only.

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.

> **Warning**
>
> **This option is available only when using Python 3.**

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.26 python-fetcher: writing fetcher-style Python sources

The Python source allows you to write your own source in Python. You can import external Python modules to receive or fetch the messages. Since many services have a Python library, the Python source makes integrating AxoSyslog very easy and quick.

You can write two different type of sources in Python:

- Server-style sources that receives messages. Write server-style sources if you want to use an event-loop based, nonblocking server framework in Python, or if you want to implement a custom loop.
- Fetcher-style sources that actively fetch messages. In general, write fetcher-style sources (for example, when using simple blocking APIs), unless you explicitly need a server-style source.

This section describes fetcher-style sources. For details on server-style sources, see python: writing server-style Python sources.

The following points apply to using Python blocks in AxoSyslog in general:

- Python parsers and template functions are available in AxoSyslog version 3.10 and later.

  Python destinations and sources are available in AxoSyslog version 3.18 and later.

- Supported Python versions: 2.7 and 3.4+ (if you are using pre-built binaries, check the dependencies of the package to find out which Python version it was compiled with).
- The Python block must be a top-level block in the AxoSyslog configuration file.
- If you store the Python code in a separate Python file and only include it in the AxoSyslog configuration file, make sure that the PYTHONPATH environment variable includes the path to the Python file, and export the PYTHON_PATH environment variable. For example, if you start AxoSyslog manually from a terminal and you store your Python files in the `/opt/syslog-ng/etc` directory, use the following command: `export PYTHONPATH=/opt/syslog-ng/etc`.

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

- The Python object is initiated every time when AxoSyslog is started or reloaded.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

- The Python block can contain multiple Python functions.
- Using Python code in AxoSyslog can significantly decrease the performance of AxoSyslog, especially if the Python code is slow. In general, the features of AxoSyslog are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The AxoSyslog application does not do any of this.
- Python error messages are available in the `internal()` source of AxoSyslog.
- You can access the name-value pairs of AxoSyslog directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of AxoSyslog. For details, see Logging from your Python code.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

Python sources consist of two parts. The first is a AxoSyslog source object that you define in your AxoSyslog configuration and use in the log path. This object references a Python class, which is the second part of the Python source. The Python class receives or fetches the log messages, and can do virtually anything that you can code in Python. You can either embed the Python class into your AxoSyslog configuration file, or store it in an external Python file .

```
source <name_of_the_python_source>{
     python-fetcher(
         class("<name_of_the_python_class_executed_by_the_source>")
     );
 };

 python {
 from syslogng import LogFetcher
 from syslogng import LogMessage

 class <name_of_the_python_class_executed_by_the_source>(LogFetcher):
     def init(self, options): # optional
         print("init")
         print(options)
         return True

     def deinit(self): # optional
         print("deinit")

     def open(self): # optional
         print("open")
         return True

     def fetch(self): # mandatory
         print("fetch")
         # return LogFetcher.FETCH_ERROR,
         # return LogFetcher.FETCH_NOT_CONNECTED,
         # return LogFetcher.FETCH_TRY_AGAIN,
         # return LogFetcher.FETCH_NO_DATA,
         return LogFetcher.FETCH_SUCCESS, msg

     def request_exit(self):
         print("request_exit")
         # If your fetching method is blocking, do something to break it
         # For example, if it reads a socket: socket.shutdown()

     def close(self): # optional
         print("close")
 };
```

Methods of the python-fetcher() source

Fetcher-style Python sources must be inherited from the `syslogng.LogFetcher` class, and must implement at least the `fetch` method. Multiple inheritance is allowed, but only for pure Python super classes.

For fetcher-style Python sources, AxoSyslog handles the event loop and the scheduling automatically. You can use simple blocking server/client libraries to receive or fetch logs.

You can retrieve messages using the `fetch()` method.

init(self, options) method (optional)

The AxoSyslog application initializes Python objects every time when it is started or reloaded. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your source to work.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

When this method returns with False, AxoSyslog does not start. It can be used to check options and return False when they prevent the successful start of the source.

`options` : This optional argument contains the contents of the `options()` parameter of the AxoSyslog configuration object as a Python dictionary.

open(self) method (optional)

The `open(self)` method opens the resources required for the source, for example, it initiates a connection to the target service. It is called after `init()` when AxoSyslog is started or reloaded. If `fetch()` returns with an error, AxoSyslog calls the `close()` and `open()` methods before trying to fetch a new message.

If `open()` fails, it should return the False value. In this case, AxoSyslog retries it every `time-reopen()` seconds. By default, this is 1 second for Python sources and destinations, the value of `time-reopen()` is not inherited from the global option. For details, see Error handling in the python() destination .

fetch(self) method (mandatory)

Use the `fetch` method to fetch messages and pass them to the log paths.

For details on parsing messages, see Python LogMessage API.

The `fetch` method must return one of the following values:

- `LogFetcher.FETCH_ERROR` : Fetching new messages failed, AxoSyslog calls the `close` and `open` methods.
- `LogFetcher.FETCH_NO_DATA` : There was not any data available. The source waits before calling the fetch method again. The wait time is equal to `time-reopen()` by default, but you can override it by setting the `fetch-no-data-delay()` option in the source.
- `LogFetcher.FETCH_NOT_CONNECTED` : Could not access the source, AxoSyslog calls the `open` method.
- `LogFetcher.FETCH_SUCCESS, msg` : Post the message returned as the second argument.
- `LogFetcher.FETCH_TRY_AGAIN` : The fetcher could not provide a message this time, but will make the source call the fetch method as soon as possible.

request_exit(self) method (optional)

If you use blocking operations within the `fetch()` method, use `request_exit()` to interrupt those operations (for example, to shut down a socket), otherwise AxoSyslog is not able to stop. Note that AxoSyslog calls the `request_exit` method from a thread different from the source thread.

close(self) method (optional)

Close the connection to the target service. Usually it is called right before `deinit()` when stopping or reloading AxoSyslog. It is also called when `fecth()` fails.

close_batch(self)

Closes the current source-side batch. Source-side batching helps AxoSyslog to effectively process a larger chunk of messages, instead of processing messages each message. For example, when feeding a destination queue and instead of taking a lock on the queue for every message (causing contention), we only take it once per batch.

The native drivers built into AxoSyslog typically close batches once every mainloop iteration, allowing a single iteration to process multiple messages. For instance, when receiving multiple messages in a single TCP datagram, all of those messages can be processed as a part of the same batch.

In Python-based log sources, a batch will automatically be closed after every message posted via `post_message()`, except if `self.auto_close_batches` is set to `False` during initialization. In case `self.auto_close_batches` is set to `False`, the driver has to call `close_batch()` explicitly, preferably at a natural boundary between incoming batches of messages. A good example is when we retrieve several messages via the same HTTP REST call, then the right time to close the batch would be after the last message in the response is posted.

The deinit(self) method (optional)

This method is executed when AxoSyslog is stopped or reloaded. This method does not return a value.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

For the list of available optional parameters, see python() and python-fetcher() source options.

7.27 qBittorrent logs

AxoSyslog can collect logs of the qBittorrent application.

Prerequisites

- AxoSyslog version 4.6.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `qbittorrent()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/qbittorrent/qbittorrent.conf on GitHub.

Configuration

```
source s_qbittorrent {
  qbittorrent(
    dir("/path/to/qbittorrent-root-log-dir")
  );
};
```

To configure the source, you only have to specify the root log directory of qBittorrent in the `dir()` parameter. You can find the root log directory by selecting **Tools > Preferences > Behavior > Log file > Save path** in the qBittorrent application. Otherwise, the `qbittorrent()` source has the same parameters as the `file()` source .

7.28 snmptrap: Read Net-SNMP traps

Using the `snmptrap()` source, you can read and parse the SNMP traps of the Net-SNMP 's `snmptrapd` application. AxoSyslog 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 AxoSyslog application automatically adds the `.snmp.` prefix to the name of the fields the extracted from the message.

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 AxoSyslog discards these messages.
- The AxoSyslog 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, AxoSyslog 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 AxoSyslog macros (name-value pairs). Therefore, the AxoSyslog 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

- AxoSyslog version 3.10 and later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"`, which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

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

- This feature requires a separate module. Install the `axosyslog-mod-snmp` package on Debian/Ubuntu, or the `axosyslog-afsnmp` package on RHEL and compatible distributions. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.
- 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 AxoSyslog expects this default format:

    ```
    ```shell
    format2 %.4y-%.2m-%.2l %.2h:%.2j:%.2k %B [%b]:\n%v\n
    ```
    ```

  - Because 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.

Configuration

To use the `snmptrap()` driver, the `scl.conf` file must be included in your AxoSyslog configuration:

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

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, AxoSyslog 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 AxoSyslog 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, AxoSyslog 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"
       }
    }
```

7.28.1 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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:	N/A

Description: If you receive the following error message during AxoSyslog 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

Default value: `.snmp.` option.

set-message-macro()

Type:	`yes` or `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 `${MESSAGE}` part of the log message. If `set-message-macro()` is set to `no` , AxoSyslog leaves the `${MESSAGE}` part empty. If `set-message-macro()` is set to `yes` , AxoSyslog generates a regular log message from the trap.

Options of the snmptrapd-parser() parser

The `snmptrap()` source reads the log file of `snmptrapd` and parses it with the `snmptrapd-parser()` parser. If you want to parse traps that arrive from somewhere else, you can also use `snmptrapd-parser()` directly in a parser statement:

```
parser p_snmptrap {
     snmptrapd-parser(prefix(".snmp."));
 };
```

Used this way, the parser has the `prefix()` and `set-message-macro()` options described above, and also the following options.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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}`).

7.29 splunk-hec: Receive messages from Splunk HEC clients

The `splunk-hec()` source receives messages from clients that send data to the Splunk HTTP Event Collector (HEC) , for example, from Splunk Connect for Syslog (SC4S) . This way you can point your existing HEC clients at AxoSyslog without reconfiguring them, and process or route the messages before forwarding them to Splunk or to another destination.

To send messages to Splunk, use the `splunk-hec-event()` destination instead.

> **Note**
>
> **This driver is a reusable configuration snippet. For details on such configuration snippets, see Reusing configuration blocks.**

Prerequisites

- AxoSyslog version 4.28 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `splunk-hec()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/splunk/splunk-src.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Example: Receiving HEC events

Minimal configuration:

```
@include "scl.conf"

source s_splunk_hec {
  splunk-hec(
    token("a1b2c3d4-e5f6-4a7b-8c9d-0e1f2a3b4c5d")
  );
};
```

To receive the events over HTTPS, configure TLS:

```
source s_splunk_hec_tls {
  splunk-hec(
    token("a1b2c3d4-e5f6-4a7b-8c9d-0e1f2a3b4c5d")
    transport(tls)
    tls(
      key-file("/etc/syslog-ng/hec.key")
      cert-file("/etc/syslog-ng/hec.crt")
    )
  );
};
```

To test the source, you can use `curl` on the host where AxoSyslog is running:

```
curl -X POST -H "Authorization: Splunk a1b2c3d4-e5f6-4a7b-8c9d-0e1f2a3b4c5d" \
  --data '{"time": 1757000000.000, "host": "example-host", "event": "hello from
HEC"}' \
  http://127.0.0.1:8088/services/collector
```

AxoSyslog accepts the request and answers it with the response the HEC clients expect:

```
{"text":"Success","code":0}
```

How the messages are processed

The `splunk-hec()` source handles the incoming requests as follows:

1. It authenticates the request: the `Authorization` header must be `Splunk <token>`, where `<token>` is the value of the `token()` option. AxoSyslog rejects every other request with `401 Unauthorized`.

2. It extracts the events from the body of the request. A request can contain a JSON array of events, or several concatenated JSON objects, and each event becomes a separate log message.

3. It parses each event with the `json-parser()`, and makes the fields available with the prefix set in the `prefix()` option, for example, `${.splunk.source}` and `${.splunk.sourcetype}`.

4. It maps the standard HEC fields to AxoSyslog macros:

HEC field	Macro
`event`	`${MESSAGE}`
`host`	`${HOST}`
`time`	The timestamp of the message. AxoSyslog accepts both the `<seconds>.<milliseconds>` and the `<seconds>` epoch formats.

   AxoSyslog maps a field only if it is present and not empty, otherwise the macro keeps its original value.

5. If the payload is not valid JSON, AxoSyslog parses it as a syslog message instead. This way the source also accepts the raw endpoint of HEC.

The source accepts `gzip` and `deflate` compressed request bodies.

splunk-hec() source options

The `splunk-hec()` source has the following options.

> **Note**
>
> **The `splunk-hec()` source sets the `mode()`, `auth-token()`, `response-body()`, and `flags()` options of the underlying `ehttp()` source. Set the `token()` option instead of `auth-token()`. Setting `mode()` or `response-body()` overrides the value the driver uses and breaks compatibility with the HEC clients.**

chain-hostnames()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option.

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no`.

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

host-override()

Type:	string
Default:	

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

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

ip() or localip()

Type:	string
Default:	`0.0.0.0`

Description: The IP address to bind to. By default, AxoSyslog listens on every available interface. Note that this is not the address where messages are accepted from.

ip-freebind()

Type:	yes or no
Default:	no

Description: Linux-only. Enables the `IP_FREEBIND` socket option, allowing AxoSyslog to bind to a non-local IP address. Useful when the address isn't yet configured on any interface (for example, during failover scenarios) so the source can listen before the address becomes available. Has no effect on platforms that don't support `IP_FREEBIND`.

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 AxoSyslog is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the AxoSyslog 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)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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:	The value of `max-connections()` multiplied by `min-iw-size-per-reader()`

Description: The size of the source window, that is, the maximum number of in-flight messages the source permits before flow control is enforced. This applies only when `flags(flow-control)` is set. For details, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string

Default:	

This option is deprecated. Use `program-override` instead.

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.

max-connections()

Type:	number
Default:	100

Description: The maximum number of simultaneous client connections that the source accepts.

max-request-size()

Type:	number [bytes]
Default:	The value of `log-msg-size()` multiplied by 1000

Description: The maximum size of a single HTTP request, in bytes. AxoSyslog rejects larger requests with `413 Payload Too Large` and logs an error.

For compressed requests, this limit applies to the decompressed body as well.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

port()

Type:	number
Default:	8088

Description: The port number to bind to. The default is the default port of the Splunk HTTP Event Collector. Setting `transport(tls)` doesn't change this default. Make sure to enable this port on the firewall of the AxoSyslog host.

prefix()

Type:	string
Default:	`.splunk.`

Description: The prefix that AxoSyslog inserts before the name of the fields parsed from the event, to avoid collisions with other name-value pairs of the message. For example, with the default prefix, the `sourcetype` field of the event becomes available as `${.splunk.sourcetype}`.

Names starting with a dot (for example, `.example`) are reserved for use by AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	`yes` or `no`
Default:	yes

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

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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.

Warning

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

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

so-sndbuf()

Type:	number
Default:	0

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

tags()

Type:	string
Default:	

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

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

Warning

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 AxoSyslog 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.

Warning

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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog version 3.4 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.

tls()

Type:	TLS options
Default:	

Description: The TLS settings of the source, for example, the certificate and the private key that AxoSyslog shows to the clients. Setting `transport(tls)` requires a `tls()` block. For details on the individual options, see TLS options.

```
tls(
    key-file("/etc/syslog-ng/server.key")
    cert-file("/etc/syslog-ng/server.crt")
)
```

The following TLS options are available: `ca-dir()`, `cert-file()`, `cipher-suite()`, `crl-dir()`, `dhparam-file()`, `ecdh-curve-list()`, `key-file()`, `peer-verify()`, `pkcs12-file()`, `ssl-options()`, `trusted-dn()`, and `trusted-keys()`.

> **Note**
>
> **This is a narrower set than what the `network()` and `syslog()` sources accept. In particular, `ca-file()` is not available, use `ca-dir()` instead. The `trusted-fingerprints()`, `ocsp-stapling-verify()`, `sigalgs()`, and `keylog-file()` options are not available either.**

token()

Type:	string
Default:	

Description: The HEC token that the clients must send in the `Authorization` header of their requests. This is the token that you would otherwise configure on your Splunk deployment, so you can reuse the token your clients already have.

This option is mandatory: if you don't set it, AxoSyslog rejects every request. Since the token is sent in clear text, use `transport(tls)` to receive the events over HTTPS.

transport()

Type:	`tcp` or `tls`
Default:	`tcp`

Description: Sets whether the source accepts plain HTTP (`tcp`) or HTTPS (`tls`) connections. Setting `transport(tls)` requires a `tls()` block, and changes the default port to 443.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.30 stdin: Collect messages from the standard input stream

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

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

Declaration

```
stdin();
```

For the list of available optional parameters, see stdin() source options.

Example: Using the stdin() driver

```
@version: 4.29
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
```

7.30.1 stdin() source options

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

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

default-facility()

Type:	facility string

Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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, AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**

Default: N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type: string

Default: N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type: string

Default: N/A

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

`teardown()`

Type: string

Default: N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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()`.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the

`syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

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, AxoSyslog 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()`, AxoSyslog 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`.

> **Warning**
>
> **If the `multi-line-garbage()` option is set, AxoSyslog discards lines between the line matching the `multi-line-garbage()` and the next line matching `multi-line-prefix()`.**

multi-line-mode()

Type:	`indented`, `prefix-garbage`, `prefix-suffix`, `regexp`, `smart`
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The AxoSyslog application provides the following methods to process multi-line messages:

- `indented`: 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.

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

- `prefix-garbage` : 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.

- `prefix-suffix` : 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.

- `smart` : The `smart` mode recognizes multi-line data backtraces even if they span multiple lines in the input. The backtraces are converted to a single log message for easier analysis. Backtraces for the following programming languages are recognized : Python, Java, JavaScript, PHP, Go, Ruby, and Dart.

  `smart` mode is available in AxoSyslog version 4.2 and newer.

  The regular expressions to recognize these programming languages are specified in an external file called `/usr/share/syslog-ng/smart-multi-line.fsm` (installation path depends on configure arguments), in a format that is described in that file.

Note

To format multi-line messages to your individual needs, consider the following:

- 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, AxoSyslog ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the `multi-line-garbage()` option.

Note

To format multi-line messages to your individual needs, consider the following:

- 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:2
5)
           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:2
5)
           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 AxoSyslog trying to interpret the date in the message.

multi-line-suffix()

Type:	regular expression
Default:	empty string

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

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

multi-line-timeout()

Type:	nonnegative integer (seconds)
Default:	0 (off)

Description: When using a multi-line mode (such as `regexp` or `indented`), sets the maximum time AxoSyslog waits for the next line of a multi-line message before flushing the partial message it has buffered. `0` disables the timeout. Set this option to avoid stuck partial messages when a producer pauses between lines.

normalize-hostnames()

Accepted values:	`yes`, `no`

Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()` , AxoSyslog will not read anymore from this pipe and displays the following error message:

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

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option. The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.31 sun-streams: Collect 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 AxoSyslog 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 AxoSyslog `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 AxoSyslog GitHub repository .**

Prerequisites

AxoSyslog supports this feature only on Solaris. The Debian/Ubuntu and RHEL packages that Axoflow publishes don't include it. To use it, build AxoSyslog from source on Solaris.

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"));
 };
```

7.31.1 sun-streams() source options

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

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.

- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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, AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

optional()

Type:	yes or no
Default:	

Description: Instruct AxoSyslog to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()` , `unix-dgram` , and `unix-stream` drivers.

pad-size()

Type:	number
Default:	0

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

7.32 syslog: Collect messages using the IETF-syslog protocol

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 AxoSyslog versions 3.1 and earlier, the `syslog()` driver could handle only messages in the IETF-syslog (RFC 5424-26) format.
>
> Starting with version 4.10, AxoSyslog can automatically detect RFC6587-style octet-count based framing. For details, see the transport option .

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')
     )
     );
 };
```

Warning

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

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

See also

- Configuring AxoSyslog as a logserver
- Configuring TLS server authentication on the AxoSyslog server
- Configuring mutual TLS authentication on the AxoSyslog server

7.32.1 syslog() source options

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

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

dynamic-window-realloc-ticks()

Type:	nonnegative number
Default:	10

Description: When `dynamic-window-size()` is enabled, controls how many statistics ticks must pass before AxoSyslog reallocates the dynamic window across active connections. Larger values dampen reallocation churn under bursty traffic; smaller values let the window adapt more aggressively to per-connection load. For background, see Managing incoming and outgoing messages with flow-control.

dynamic-window-size()

Type:	number
Default:	0

Description: The size of the dynamic control window used during flow-control. For details on flow-control, see Dynamic flow-control.

dynamic-window-stats-freq()

Type:	nonnegative float (seconds)
Default:	0.0 (off)

Description: When `dynamic-window-size()` is set to a non-zero value, sets how often AxoSyslog samples per-connection throughput statistics that drive window reallocation. `0` turns off sampling. The interval is stored internally in milliseconds, so fractional seconds are honored.

ebpf()

Available in AxoSyslog version 4.2 and newer.

By default, the kernel chooses the receive socket for a specific UDP randomly based on the source IP/port of the sender. You can customize this algorithm using the Extended Berkeley Packet Filter (eBPF) plugin. The `ebpf()` option changes the kernel's SO_REUSEPORT algorithm so that all messages are randomly placed into one of the UDP sockets. The decision which UDP socket buffer a datagram is placed is made for every datagram, and not once for every stream. This means that messages are perfectly load-balanced across your set of UDP sockets. While this resolves the imbalance between the sockets and results in perfect load balancing, you will lose ordering between messages from the same sender, which is the price to pay for increased throughput.

```
source s_network {
    network(
        transport("udp")
        so-reuseport(1) persist-name("udp1")
        ebpf(reuseport(sockets(4)))
    );
    network(transport("udp") so-reuseport(1) persist-name("udp2"));
    network(transport("udp") so-reuseport(1) persist-name("udp3"));
    network(transport("udp") so-reuseport(1) persist-name("udp4"));
};
```

For a detailed introduction, see the Scaling syslog to 1M EPS with eBPF blog post.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)

- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

- *threaded*: The `threaded` flag enables multithreading for the source. For details on multithreading, see Multithreading and scaling.

> **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.

> **Note**
>
> **The AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

interface()

Type:	string
Default:	None

Description: Bind to the specified interface instead of an IP address. Available in 3.19 and later.

ip() or localip()

Type:	string
Default:	0.0.0.0

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

ip-freebind()

Type:	yes or no
Default:	no

Description: Linux-only. Enables the `IP_FREEBIND` socket option, allowing AxoSyslog to bind to a non-local IP address. Useful when the address isn't yet configured on any interface (for example, during failover scenarios) so the source can listen before the address becomes available. Has no effect on platforms that don't support `IP_FREEBIND`.

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 AxoSyslog is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the AxoSyslog 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)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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)
         );
 };
```

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-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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

If the `max-connections()` option is set, the `log-iw-size()` will be divided by the number of connections (except for UDP-based connection), 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 AxoSyslog clients, make sure that the window size is larger than the `flush-lines()` option set in the destination of your clients.

Example: Initial window size of a connection

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

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-connections()

Type:	number

Default:	10

Description: Specifies the maximum number of simultaneous connections. Doesn't affect UDP-based traffic, which is always treated as a single connection.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()` , AxoSyslog will not read anymore from this pipe and displays the following error message:

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

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

port() or localport()

Type:	number
Default:	In case of TCP transport: 514 In case of UDP transport: 514

Description: The port number to bind to.

program-override()

Type:	string
Default:	

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

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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.

> **Warning**
>
> When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the AxoSyslog server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In this cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.
>
> As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.

so-reuseport()

Type:	yes or no
Default:	no

Description: Enables SO_REUSEPORT on systems that support it. When enabled, the kernel allows multiple UDP sockets to be bound to the same port, and the kernel load-balances incoming UDP datagrams to the sockets. The sockets are distributed based on the hash of (srcip, dstip, srcport, dstport), so the same listener should be receiving packets from the same endpoint. For example:

```
   source {
            network(transport("udp") so-reuseport(1) port(2000) persist-
name("udp1"));
            network(so-reuseport(1) port(2000) persist-name("udp2"));
            network(so-reuseport(1) port(2000) persist-name("udp3"));
            network(so-reuseport(1) port(2000) persist-name("udp4"));
    };
```

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-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 version 3.1 and later.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

> **Warning**
>
> 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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog version 3.4 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:	`auto`, `nul-terminated`, `proxied-tcp`, `proxied-tls`, `proxied-tls-passthrough`, `proxied-udp`, `tcp`, `text-with-nuls`, `tls`, `udp`
Default:	`tcp`

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

- `auto`: Available in version 4.10 and later. Automatically detects RFC6587-style octet-count based framing in TCP traffic. In future versions this functionality will be extended to cover and detect all TCP-based protocols.
- `nul-terminated`: Available in version 4.21 and later. Uses NUL characters to separate log records instead of newline characters. For example, the following message body will be interpreted as a multi-line message: `line1\nline2\nline3\x00`

- `proxied-tcp` , `proxied-tls` , `proxied-tls-passthrough` , `proxied-udp` : Refers to the HAProxy Proxy Protocol. `proxied-udp` is available in version 4.25 and later. For details, see Proxy Protocol support.

- `text-with-nuls` : Allows embedded `NUL` characters in the message from a TCP source, that is, AxoSyslog will not delimiter the incoming messages on `NUL` characters, only on `newline` characters (contrary to `tcp` transport, which splits the incoming log on `newline` characters and `NUL` characters).

> **Note**
>
> **The AxoSyslog application does not support embedded `NUL` characters everywhere, so it is recommended that you also use `flags(no-multi-line)` that causes `NUL` characters to be replaced by space.**

> **Warning**
>
> When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the AxoSyslog server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In this cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.
>
> As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.

trim-large-messages()

Type:	`yes` or `no`
Default:	Use the global `trim-large-messages()` option, which defaults to `no` .

Description: Determines what AxoSyslog does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()` . Other drivers ignore the `trim-large-messages()` option.

- If set to `no` , AxoSyslog drops the incoming log message.

- If set to `yes`, AxoSyslog trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter f_trimmed {
      tags("trimmed");
  };
  ```

  If AxoSyslog trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in AxoSyslog version 3.21 and later.

Uses the value of the global option if not specified.

tls()

Type:	tls options
Default:	n/a

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

use-dns()

Type:	`yes`, `no`, `persist_only`
Default:	`yes`

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option. The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network

endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.33 system: Collect the system-specific log messages of a platform

Starting with version 3.2, AxoSyslog 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 AxoSyslog configuration file, AxoSyslog automatically adds the following sources to the AxoSyslog configuration.

> **Note**
>
> **AxoSyslog 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 AxoSyslog.**

The `system()` driver is also used in the default configuration file of AxoSyslog. For details on the default configuration file, see Example: The default configuration file of AxoSyslog . Starting with AxoSyslog 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.

> **Warning**
>
> **If AxoSyslog does not recognize the platform it is installed on, it does not add any sources.**

Starting with version 3.6, AxoSyslog 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).

Source aut

Platform	Message source
AIX	`unix-dgram("/dev/log");`
FreeBSD	`unix-dgram("/var/run/log");` `unix-dgram("/var/run/logpriv" perm(0600));` `file("/dev/klog" follow-freq(0) program-override("kernel") flags(no-parse));` For FreeBSD versions earlier than 9.1, `follow-freq(1)` is used.
GNU/kFreeBSD	`unix-dgram("/var/run/log");` `file("/dev/klog" follow-freq(0) program-override("kernel"));`
HP-UX	`pipe("/dev/log" pad-size(2048));`

Platform	Message source
AIX	`unix-dgram("/dev/log");`
FreeBSD	`unix-dgram("/var/run/log");` `unix-dgram("/var/run/logpriv" perm(0600));` `file("/dev/klog" follow-freq(0) program-override("kernel") flags(no-parse));` For FreeBSD versions earlier than 9.1, `follow-freq(1)` is used.
GNU/kFreeBSD	`unix-dgram("/var/run/log");` `file("/dev/klog" follow-freq(0) program-override("kernel"));`
HP-UX	`pipe("/dev/log" pad-size(2048));`

Linux

Note that on Linux, the `so-rcvbuf()` option of the `system()` source is automatically set to 8192.

If the host is running under systemd, AxoSyslog reads both syslog and kernel messages directly from the systemd journal file using the `systemd-journal()` source. In this case, AxoSyslog doesn't read from `/dev/log` nor `/proc/kmsg`.

If the kernel of the host is version 3.5 or newer, and `/dev/kmsg` is seekable, AxoSyslog will use that instead of `/proc/kmsg`, using the `multi-line-mode(indented)`, `keep-timestamp(no)`, and the `format(linux-kmsg)` options.

If AxoSyslog is running in a jail or a Linux Container (LXC), it will not read from the `/dev/kmsg` or `/proc/kmsg` files.

- With systemd:

  ```
  systemd-journal();
  ```

- Without systemd, on kernel 3.5 or newer:

  ```
  unix-dgram("/dev/log"); file("/dev/kmsg" program-override("kernel") flags(kernel) format("linux-kmsg") keep-timestamp(no));
  ```

- Without systemd, on kernels older than 3.5:

  ```
  unix-dgram("/dev/log"); file("/proc/kmsg" program-override("kernel") flags(kernel) keep-timestamp(no));
  ```

macOS

```
file("/var/log/system.log" follow-freq(1));
```

> **Note**
>
> **Starting with version 3.7, the AxoSyslog `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 AxoSyslog GitHub repository .**

NetBSD

```
unix-dgram("/var/run/log");
```

> **Note**
>
> Starting with version 3.7, the AxoSyslog `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 AxoSyslog GitHub repository .

OpenBSD

```
openbsd();
```

Solaris 8

```
sun-streams("/dev/log");
```

> **Note**
>
> Starting with version 3.7, the AxoSyslog `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 AxoSyslog GitHub repository .

Solaris 9

```
sun-streams("/dev/log" door("/etc/.syslog_door"));
```

> **Note**
>
> Starting with version 3.7, the AxoSyslog `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 AxoSyslog GitHub repository .

Solaris 10 `sun-streams("/dev/log" door("/var/run/syslog_door"));`

> **Note**
>
> **Starting with version 3.7, the AxoSyslog `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 AxoSyslog GitHub repository .**

`system()` source options

The `system()` 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

7.34 systemd-journal: Collect 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, AxoSyslog adds the `.journald.` prefix to the name of every parsed value. For a list and description of name-value pairs that journald provides, see the documentation of journald for your platform (for example, `man systemd.journal-fields`).

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

Warning

Only one `systemd-journal()` source can be configured in the configuration file. If there is more than one `systemd-journal()` source configured, AxoSyslog will not start.

Declaration:

```
systemd-journal(options);
```

If you want to use multiple `systemd-journal()` sources in your configuration, the sources must use unique systemd namespaces. For details, see the `namespace()` option .

Example: Send all fields through syslog protocol

To send all fields through the syslog protocol, enter the prefix in the following format: “ `.SDATA.<name>` ”.

```
@version: 4.29

 source s_journald {
     systemd-journal(prefix(".SDATA.journald."));
 };

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

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

Example: Filter for a specific field

```
   @version: 4.29

    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: Send all fields in value-pairs

```
   @version: 4.29

    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 AxoSyslog application makes this information available in the following macros:

Journald fields as macros

Journald field	AxoSyslog predefined macro
MESSAGE	$MESSAGE
_HOSTNAME	$HOST
_PID	$PID
_COMM or SYSLOG_IDENTIFIER	$PROGRAM If both _COMM and SYSLOG_IDENTIFIER exists, AxoSyslog uses SYSLOG_IDENTIFIER
SYSLOG_FACILITY	$FACILITY_NUM
PRIORITY	$LEVEL_NUM

7.34.1 systemd-journal() source options

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

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

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-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog will retain the hostname information read from the systemd journal messages.
- If disabled (`keep-hostname(no)`), AxoSyslog will use the hostname that has been set up for the operating system instance that AxoSyslog 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.

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

match-boot()

Type:	`yes` , `no`
Default:	no

Available in AxoSyslog 4.1 and later.

Description: If set to yes, AxoSyslog fetches only journal messages that relate to the current boot, and to ignores messages generated in previous boots.

matches()

Type:	arrow list
Default:	

Available in AxoSyslog 4.1 and later.

Description: Specifies one or more filters to apply on the journal fields, similarly how you can use `journalctl` . For example:

```
matches(
    "_COMM" => "systemd"
    )
```

max-field-size()

Type:	number (characters)
Default:	65536

Description: The maximum length of a field's value.

namespace()

Type:	string
Default:	“*”

Description: The `namespace()` option works exactly the same way as the respective option of the Journalctl command line tool .

The following modes of operation are available:

- If you do not specify the `namespace()` option in your configuration, or if you specify an empty string, the `systemd-journal()` source reads and displays log data from all namespaces.
- If you specify the `namespace()` option as `namespace("*")`, the `systemd-journal()` source reads and displays log data from all namespaces, interleaved.
- If `namespace(<specific-namespace>)` is specified, the `systemd-journal()` source only reads and displays log data from the specified namespace.
- If the namespace identifier is prefixed with `"+"` when you specify your `namespace()` option, the `systemd-journal()` source only reads and displays log data from the specified namespace and the default namespace, interleaved.

Starting with AxoSyslog 4.4, you can use multiple `systemd-journal()` sources in your configuration. In this case, each source must use unique systemd namespaces.

Syntax: `namespace(string)`

Example: configuration examples for using the namespace() option

The following configuration example uses the default value for the `namespace()` option:

```
source s_journal
 {
   systemd-journal(namespace("*"));
 };
```

The following configuration example uses a prefixed namespace identifier in the `namespace()` option:

```
source s_journal
 {
   systemd-journal(namespace("+foobar"));
 };
```

> **Note**
>
> **Namespace support was introduced to the Journalctl command line tool in Systemd version 2.45. The AxoSyslog application supports the `namespace()` option from version 3.29. For further information about namespaces on the Systemd side, see Journal Namespaces .**

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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()

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, AxoSyslog adds the `.journald.` prefix to every value.

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	`yes` or `no`
Default:	yes

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

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.35 systemd-syslog: Collect 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 AxoSyslog 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: 4.29

 source s_systemdd {
     systemd-syslog();
 };

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

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

7.35.1 systemd-syslog() source options

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

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no

Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)`.

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`

- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()` , the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

normalize-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

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` .

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.` , using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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.

Warning

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

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

so-reuseport()

Type:	yes or no
Default:	no

Description: Enables SO_REUSEPORT on systems that support it. When enabled, the kernel allows multiple UDP sockets to be bound to the same port, and the kernel load-balances incoming UDP datagrams to the sockets. The sockets are distributed based on the hash of (srcip, dstip, srcport, dstport), so the same listener should be receiving packets from the same endpoint. For example:

```
   source {
            network(transport("udp") so-reuseport(1) port(2000) persist-
name("udp1"));
            network(so-reuseport(1) port(2000) persist-name("udp2"));
            network(so-reuseport(1) port(2000) persist-name("udp3"));
            network(so-reuseport(1) port(2000) persist-name("udp4"));
    };
```

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-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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option. The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes`, `no`

Default: `no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.36 tcp, tcp6, udp, udp6: OBSOLETE - Collect messages from remote hosts using the BSD syslog protocol

> **Note**
>
> **The `tcp()`, `tcp6()`, `udp()`, and `udp6()` drivers are obsolete. Use the `network()` source and the `network()` destination instead. For details, see network: Collect messages using the RFC3164 protocol (network() driver) and network: Send 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.

7.36.1 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: Collect messages using the RFC3164 protocol (network() driver) and network: Send 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.

7.36.1.1 Change an old source driver to the network() driver

To replace your existing `tcp()`, `tcp6()`, `udp()`, `udp6()` sources with a `network()` source, complete the following steps.

1. Replace the driver with `network`. For example, replace `udp(` with `network(`
2. Set the transport protocol.
   - If you used TLS-encryption, add the `transport("tls")` option, then continue with the next step.
   - If you used the `tcp` or `tcp6` driver, add the `transport("tcp")` option.
   - If you used the `udp` or `udp` driver, add the `transport("udp")` option.
3. If you use IPv6 (that is, the `udp6` or `tcp6` driver), add the `ip-protocol(6)` option.
4. If you did not specify the port used in the old driver, check 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')
        )
    );
};
```

7.37 unix-stream, unix-dgram: Collect messages from UNIX domain sockets

The `unix-stream()` and `unix-dgram()` drivers open an `AF_UNIX` socket and start listening on it for messages. On Linux both the `unix-stream()` and `unix-dgram()` drivers are used and are always reliable. The `unix-stream()` driver uses `SOCK_STREAM` semantics (connection oriented), while `unix-dgram()` is used on BSDs and uses `SOCK_DGRAM` semantics.

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");
 };
```

7.37.1 UNIX credentials and other metadata

Starting with AxoSyslog 3.6, the `unix-stream()` and `unix-dgram()` sources automatically extract the available UNIX credentials and other metainformation from the received log messages. The AxoSyslog 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.

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 credentials available via UNIX domain sockets

7.37.2 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:

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

create-dirs()

Type:	yes or no
Default:	no

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

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)`.

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

dynamic-window-realloc-ticks()

Type:	nonnegative number
Default:	10

Description: When `dynamic-window-size()` is enabled, controls how many statistics ticks must pass before AxoSyslog reallocates the dynamic window across active connections. Larger values dampen reallocation churn under bursty traffic; smaller values let the window adapt more aggressively to per-connection load. For background, see Managing incoming and outgoing messages with flow-control.

dynamic-window-size()

Type:	number
Default:	0

Description: The size of the dynamic control window used during flow-control. For details on flow-control, see Dynamic flow-control.

dynamic-window-stats-freq()

Type:	nonnegative float (seconds)
Default:	0.0 (off)

Description: When `dynamic-window-size()` is set to a non-zero value, sets how often AxoSyslog samples per-connection throughput statistics that drive window reallocation. `0` turns off sampling. The interval is stored internally in milliseconds, so fractional seconds are honored.

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` , `check-hostname` , `check-program` , `dont-store-legacy-msghdr` , `empty-lines` , `exit-on-eof` , `expect-hostname` , `guess-timezone` , `ignore-aux-data` , `kernel` , `no-header` , `no-hostname` , `no-multi-line` , `no-parse` , `sanitize-utf8` , `store-legacy-msghdr` , `store-raw-message` , `syslog-protocol` , `validate-utf8`
Default:	empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.
- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp,

host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-connections()

Type:	number (simultaneous connections)
Default:	256

Description: Limits the number of simultaneously open connections. Cannot be used with `unix-dgram()`.

normalize-hostnames()

Accepted values:	`yes`, `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

optional()

Type:	yes or no
Default:	

Description: Instruct AxoSyslog to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

owner()

Type:	string

Default:	root

Description: Set the uid of the socket.

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()`, AxoSyslog will not read anymore from this pipe and displays the following error message:

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

pass-unix-credentials() (DEPRECATED)

Obsolete. Use the `pass-unix-credentials()` global option, or so-passcred() per source.

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`.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

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` .

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.` , using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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-passcred()

Type:	yes or no
Default:	yes

Description: Enables the `SO_PASSCRED` socket option on UNIX domain sockets, which causes AxoSyslog to receive UNIX credentials (PID, UID, GID) of the sender process for each datagram. The credentials populate the `${.unix.pid}`, `${.unix.uid}`, and `${.unix.gid}` name-value pairs. On BSD systems, this option is also accessible under the legacy alias `local-creds()`.

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.

> **Warning**
>
> When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the AxoSyslog server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so-rcvbuf()` option of the source is increased. In this cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.
>
> As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.

so-reuseport()

Type:	yes or no
Default:	no

Description: Enables SO_REUSEPORT on systems that support it. When enabled, the kernel allows multiple UDP sockets to be bound to the same port, and the kernel load-balances incoming UDP datagrams to the sockets. The sockets are distributed based on the hash of (srcip, dstip, srcport, dstport), so the same listener should be receiving packets from the same endpoint. For example:

```
   source {
            network(transport("udp") so-reuseport(1) port(2000) persist-
name("udp1"));
            network(so-reuseport(1) port(2000) persist-name("udp2"));
            network(so-reuseport(1) port(2000) persist-name("udp3"));
            network(so-reuseport(1) port(2000) persist-name("udp4"));
    };
```

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-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 version 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.

trim-large-messages()

Type:	`yes` or `no`

Default:	Use the global `trim-large-messages()` option, which defaults to `no`.

Description: Determines what AxoSyslog does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, AxoSyslog drops the incoming log message.
- If set to `yes`, AxoSyslog trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

  ```
  filter f_trimmed {
      tags("trimmed");
  };
  ```

  If AxoSyslog trims a log message, it sends a debug-level log message to its `internal()` source.

  As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in AxoSyslog version 3.21 and later.

Uses the value of the global option if not specified.

use-dns()

Type:	`yes`, `no`, `persist_only`
Default:	`yes`

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option. The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

7.38 Webhook

AxoSyslog can collect logs via a webhook using the `webhook()` and `webhook-json()` sources. The `webhook-json()` source automatically parses the payload using the `json-parser()` .

Prerequisites

- AxoSyslog version 4.8.0 and later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `webhook()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in modules/python-modules/syslogng/modules/webhook/scl/webhook.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Configuration

Example minimal config:

```
source s_webhook {
webhook-json(
    port(8181)
    paths(["/events","/events/(?P<HOST>.*)"])
    );
};
```

To test the source, you can use `curl` for example, on the host where AxoSyslog is running:

```
curl -X POST --data "{'MESSAGE':'message-value'}" http://127.0.0.1:8181/events
```

Query parameters

If the sender includes query parameters in the URL, AxoSyslog automatically makes them available as `${webhook.query.*}`. For example, if you send data like this:

```
http://127.0.0.1:8181/events?param1=value1&param2=value2&param3=value3
```

Then the values will be available as `${webhook.query.param1}`, `${webhook.query.param2}`, `${webhook.query.param3}`.

HTTPS webhook

To receive data using HTTPS, configure a certificate and a private key for the webhook using the `tls_cert_file` and `tls_key_file` options. (Note that setting `tls_key_file` automatically changes the default port from `80` to `443`.)

To verify the certificate of the clients sending data to the webhook, set `tls_peer_verify(yes)`, and use the following options:

- `tls_use_system_cert_store(no)`, or
- `tls_ca_file("")`, or
- `tls_ca_dir("")`

Options

auth_token()

Type:	string
Default:	

Description: You can require an authentication token from the clients as an additional layer of authentication. Do not use this under plain HTTP. When `auth_token("<token>")` is set, AxoSyslog will only accept requests that contain the `Authorization: Basic <token>`, `Authorization: Bearer <token>`, or a similar header. Other requests will be rejected with 403.

For example:

```
auth_token("dGVzdF9zZWdlskfoe0aF90b2tlbg==")
```

include_request_headers()

Type:	`yes` or `no`
Default:	`no`

Available in version 4.11 and later.

Description: If enabled, the HTTP request headers from the webhook will available for processing as a JSON object under the `${webhook.headers}` field. This works for both versions of the source (`webhook()` and `webhook-json`).

paths()

Type:	JSON list
Default:	`/.*`

Description: Sets the endpoints where the webhook will receive data. You can use static paths, or regular expressions. In regular expressions you can use named capture groups to automatically set macro values.

For example, the `/events/(?P<HOST>.*)` path sets the hostname for the data received in the request based on the second part of the URL: a request to the `/events/my-example-host` URL sets the host field of that message to `my-example-host`.

You can set multiple endpoints, for example, `paths(["/events","/events/(?P<HOST>.*)"])`

port()

Type:	integer
Default:	See description

Description: The port number where the webhook is listening on, for example, `8080`. Make sure to enable the port you've configured on the firewall of the AxoSyslog host. The default value is `80` for HTTP webhooks, and `443` for HTTPS webhooks.

prefix()

Type:	string
Default:	

Description: Insert a prefix before the name part of the parsed name-value pairs to help further processing when using the `webhook-json()` source. For example, to insert the `webhook.` prefix, use the `prefix(webhook.)` option.

Names starting with a dot (for example, `.example`) are reserved for use by AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

proxy_header()

Type:	string
Default:	

Available in version 4.11 and later.

Description: By default, AxoSyslog assumes the client is sending data directly, without a proxy, and sets the `${SOURCEIP}` and `${SOURCEPORT}` macros to the IP and port of the peer. When `proxy_header()` is set:

- `${SOURCEIP}` and `${SOURCEPORT}` will be set according to the proxy header defined in `proxy_header()`, and
- the `${PEERIP}` and `${PEERPORT}` macros will contain the IP and port of the proxy.

For example, to get the proxy data from the `x-forwarded-for` request header, use:

```
webhook(port(8080) proxy-header("x-forwarded-for"));
```

and include the header in the request:

```
curl  -H "X-Forwarded-FOR: 1.2.3.4" -X POST --data "{}" http://127.0.0.1:8080/
```

Note that AxoSyslog trusts only the header that is specified in the `proxy_header()` option. If the request includes multiple headers with the specified name, the last one is used.

tls_ca_dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `tls_ca_file()` option.

tls_ca_file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
tls_ca_file("/etc/pki/tls/certs/ca-bundle.crt")
```

tls_cert_file()

Type:	filename
Default:	

Description: For HTTPS endpoints, set the `tls_cert_file` and `tls_key_file` options. Set `tls_cert_file` to the 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 `tls_key_file()` option. The AxoSyslog application shows this certificate to the clients sending data to the webhook endpoints. 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.

tls_key_file()

Type:	filename
Default:	

Description: The name of a file that contains an unencrypted private key in PEM format, suitable as a TLS key. If properly configured, the AxoSyslog application uses this private key with the matching certificate (set in the `tls_cert_file()` option).

tls_peer_verify()

Accepted values:	`yes` or `no`
Default:	`no`

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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

tls_use_system_cert_store()

Type:	`yes` or `no`
Default:	`no`

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation .

7.39 wildcard-file: Collect 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 AxoSyslog version 3.10 and later.

The AxoSyslog 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 AxoSyslog 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 how you installed AxoSyslog, 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, AxoSyslog 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 AxoSyslog 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 AxoSyslog actually monitor. The default value of `max-files()` is 100.

- If the message does not have a proper syslog header, AxoSyslog 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 AxoSyslog 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.

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)
     );
 };
```

7.39.1 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() .

> **Warning**
>
> 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)
     );
 };
```

chain-hostnames()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable the chained hostname format. For details, see the chain-hostnames() global option .

check-hostname()

Type:	boolean (`yes` or `no`)
Default:	Use the global `check-hostname()` option, which defaults to `no` .

Description:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

Uses the value of the global option if not specified.

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-level()

Type:	string
Default:	notice

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

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

dns-cache()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable DNS cache usage.

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 .

exclude-pattern()

Type:	filename without path
Default:	

Available in AxoSyslog version 4.15 and later.

Description: Can be used together with `filename-pattern()` to exclude some matching files. For example:

```
wildcard-file(
  base-dir("<pathname>")
  file-pattern("*.log");
  exclude-pattern("*.?.log");
);
```

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

Warning

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`, `check-hostname`, `check-program`, `dont-store-legacy-msghdr`, `empty-lines`, `exit-on-eof`, `expect-hostname`, `guess-timezone`, `ignore-aux-data`, `kernel`, `no-header`, `no-hostname`, `no-multi-line`, `no-parse`, `sanitize-utf8`, `store-legacy-msghdr`, `store-raw-message`, `syslog-protocol`, `validate-utf8`

Default: empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.

- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

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, AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

idle-timeout()

Accepted values:	number [seconds]
Default:	`0` (disabled)

Available in AxoSyslog 4.9 and later.

If set, AxoSyslog closes the client connection if no data is received for the specified amount of time (in seconds).

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

keep-hostname()

Type:	yes or no
Default:	no

Description: Enable or disable hostname rewriting.

- If enabled (`keep-hostname(yes)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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()`.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the

`syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

When using wildcards in the filenames, AxoSyslog 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, AxoSyslog automatically sets `log-iw-size()` to `max-files()*100` .

Example: Initial window size of file sources

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

log-msg-size()

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

Description: Maximum length of an incoming 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 (256 MiB).

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

Note

In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.

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

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

Uses the value of the global option if not specified.

log-prefix() (DEPRECATED)

Type:	string
Default:	

This option is deprecated. Use `program-override` instead.

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.

max-files()

Type:	integer
Default:	100

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

When using wildcards, AxoSyslog 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 AxoSyslog 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 AxoSyslog actually monitor. The default value of `max-files()` is 100.

monitor-method()

Type:	`auto` , `inotify` , `poll`
Default:	`auto`

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

> **Warning**
>
> **If the `multi-line-garbage()` option is set, AxoSyslog discards lines between the line matching the `multi-line-garbage()` and the next line matching `multi-line-prefix()`.**

multi-line-mode()

Type:	`indented`, `prefix-garbage`, `prefix-suffix`, `regexp`, `smart`
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The AxoSyslog application provides the following methods to process multi-line messages:

- `indented`: 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.

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

- `prefix-garbage` : 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.

- `prefix-suffix` : 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.

- `smart` : The `smart` mode recognizes multi-line data backtraces even if they span multiple lines in the input. The backtraces are converted to a single log message for easier analysis. Backtraces for the following programming languages are recognized : Python, Java, JavaScript, PHP, Go, Ruby, and Dart.

  `smart` mode is available in AxoSyslog version 4.2 and newer.

  The regular expressions to recognize these programming languages are specified in an external file called `/usr/share/syslog-ng/smart-multi-line.fsm` (installation path depends on configure arguments), in a format that is described in that file.

Note

To format multi-line messages to your individual needs, consider the following:

- 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, AxoSyslog ignores newline characters from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the `multi-line-garbage()` option.

Note

To format multi-line messages to your individual needs, consider the following:

- 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:2
5)
           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:2
5)
           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 AxoSyslog trying to interpret the date in the message.

multi-line-suffix()

Type:	regular expression
Default:	empty string

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

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

multi-line-timeout()

Type:	nonnegative integer (seconds)
Default:	0 (off)

Description: When using a multi-line mode (such as `regexp` or `indented`), sets the maximum time AxoSyslog waits for the next line of a multi-line message before flushing the partial message it has buffered. `0` disables the timeout. Set this option to avoid stuck partial messages when a producer pauses between lines.

normalize-hostnames()

Accepted values:	`yes`, `no`

Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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.**

pad-size()

Type:	number
Default:	0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. The AxoSyslog 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()` , AxoSyslog will not read anymore from this pipe and displays the following error message:

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

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

program-override()

Type:	string
Default:	

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

read-old-records()

Type:	`yes` or `no`
Default:	yes

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

recursive()

Type:	`yes` or `no`
Default:	`no`

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

> **Warning**
>
> 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)
    );
};
```

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.`, using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

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 version 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.

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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-syslogng-pid()

Type:	`yes` , `no`
Default:	`no`

Description: When enabled, AxoSyslog overrides the PID from the original message with the PID of the `syslog-ng` process.

8 destination: Forward, send, and store log messages

Configure destination drivers to send messages to files, remote syslog servers, databases, message queues, cloud services, and observability platforms.

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.

> **Note**
>
> **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.conf` configuration file using the following syntax:

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

Note the following points:

- 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, AxoSyslog 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.

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));
 };
```

The following destination drivers are available in AxoSyslog. If these destinations do not satisfy your needs, you can extend AxoSyslog and write your own destination, for example, in C, Java, or Python. For details, see Write your own custom destination in Java or Python.

8.1 amqp: Publish messages using AMQP

The `amqp()` driver publishes messages using the AMQP (Advanced Message Queuing Protocol) . AxoSyslog supports AMQP versions 0.9.1 and 1.0. The AxoSyslog `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.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-amqp` package on Debian/Ubuntu , or the `axosyslog-amqp` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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")
         )
     );
 };
```

8.1.1 amqp() destination options

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

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

auth-method()

Accepted values:	`plain` or `external`
Default:	plain

Description: The `amqp()` driver supports the following types of authentication:

- plain: Authentication happens using username and password. This is the default.
- external: Authentication happens using an out-of-band mechanism, for example, x509 certificate peer verification, client IP address range, or similar. For more information, see the RabbitMQ documentation .

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

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 AxoSyslog 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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

exchange()

Type:	string
Default:	syslog

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

exchange-declare()

Type:	`yes` or `no`
Default:	no

Description: By default, AxoSyslog 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.

frame-size()

Type:	integer
Default:	

Description: Sets maximal frame size (the frame-max option described in the AMQP Reference Guide .

heartbeat()

Type:	number [seconds]
Default:	0 (disabled)

Description: If enabled, the AxoSyslog amqp destination sends heartbeat messages to the server periodically. During negotiation, both the amqp server and the client provide a heartbeat parameter, and the smaller is chosen for heartbeat interval. For example:

```
destination { amqp(
     vhost("/")
     exchange("logs")
     body("hello world")
     heartbeat(10)
     username(guest) password(guest)
     );
 };
```

Available in AxoSyslog version 3.21 and later.

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

max-channel()

Type:	integer
Default:	

Description: Sets maximal number of channels (the channel-max option described in the AMQP Reference Guide .

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` or `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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog 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` or `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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

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.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.

8.2 Send data to Apache Arrow Flight

The `arrow-flight()` destination can send data to an Apache Arrow Flight server over gRPC. Use this destination to deliver structured, columnar data to systems that accept Arrow Flight streams.

Prerequisites

- AxoSyslog version 4.26.0 or later.
- The `arrow-flight()` destination is available in a separate module. Install the `axosyslog-mod-arrow-flight` package on Debian/Ubuntu . Currently the module isn't available for RHEL and compatible distributions , so it isn't available in the RPM packages.

  If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- An Apache Arrow Flight server that accepts the schema you configure.

Configuration

You define the columns and their types using the `schema()` option, and map each column to an AxoSyslog template or macro that is evaluated for every message routed to the destination. For example:

```
destination d_arrow_flight {
  arrow-flight(
    url("grpc://flight.example.com:8815")
    path("events.${HOST}")
    schema(
      "timestamp" TIMESTAMP => "${UNIXTIME}"
      "host"      STRING    => "${HOST}"
      "program"   STRING    => "${PROGRAM}"
      "severity"  INT64     => "${LEVEL_NUM}"
      "message"   STRING    => "${MSG}"
    )
    workers(4)
  );
};
```

Options

This destination has the following options:

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-bytes()

Type:	number [bytes]
Default:	0 (disabled)

Available in AxoSyslog version 4.26.0 and later.

Description: Sets the maximum size of the payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option. Setting `batch-bytes()` without setting `batch-lines()` removes the limit on the number of messages in a batch.

batch-lines()

Type:	number
Default:	0

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

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:

capacity-bytes()

Type:	number (bytes)
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

Note

Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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-alive()

Configures how AxoSyslog sends gRPC keepalive pings .

max-pings-without-data()

Type:	integer
Default:	

Description: The maximum number of gRPC pings that can be sent when there is no data/header frame to be sent. AxoSyslog won't send any pings after this limit. Set it to 0 disable this restriction and keep sending pings.

time()

Type:	number [milliseconds]
Default:	

Description: The period (in milliseconds) after which AxoSyslog sends a gRPC keepalive ping.

timeout()

Type:	number [milliseconds]
Default:	10

Description: The time (in milliseconds) AxoSyslog waits for an acknowledgement.

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.

path()

Type:	template
Default:	

Description: The descriptor path that identifies the dataset (for example, a table) on the Flight server. The `path()` option is templatable, so a single destination can route messages to multiple datasets based on the content of each message. For example:

```
path("events.${HOST}")
```

When `path()` resolves to different values for different messages, AxoSyslog uses it as the default value of the `worker-partition-key()` option, so that each batch contains messages belonging to the same path.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()` , AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

schema()

Type:	arrow list
Default:	

Description: Sets the columns of the data sent to the Flight server. This option is required. On the left side of the arrow (`=>`), set the name of the column and its type. On the right side, set any AxoSyslog template or macro, which is evaluated for each message routed to the destination. For example:

```
schema(
  "timestamp" TIMESTAMP => "${UNIXTIME}"
  "host"      STRING    => "${HOST}"
  "severity"  INT64     => "${LEVEL_NUM}"
  "message"   STRING    => "${MSG}"
  "labels"    MAP(STRING STRING) => "${MSG}"
)
```

The following column types are available:

Type	Aliases	Description
`STRING`		UTF-8 string. This is the default type if you omit the type keyword.
`INT64`	`INTEGER`	64-bit signed integer.
`DOUBLE`	`FLOAT64`	Double-precision floating-point number.
`BOOL`	`BOOLEAN`	Boolean value.
`TIMESTAMP`		Timestamp value.
`MAP(STRING STRING)`		Map with string keys and string values.

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.

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() (DEPRECATED)

Type:	number [seconds]
Default:	0 (no timeout)

Available in AxoSyslog version 4.26.0, deprecated in 4.27.

Description: Use `keep-alive()` instead.

url()

Type:	string
Default:	

Description: The address of the Apache Arrow Flight server, for example, `grpc://flight.example.com:8815`. If you omit the scheme from the URL (like `flight.example.com:8815`), AxoSyslog assumes `grpc://`.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

worker-partition-autoscaling()

Type:	`yes` , `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)` , set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()` . Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()` .

worker-partition-key()

Type:	template
Default:	value of `path()`

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled and multiple `workers()` are configured, only messages that belong to the same `path()` should be added to the same batch. By default, AxoSyslog uses the `path()` template as the partition key, so you only need to set `worker-partition-key()` explicitly if you want to partition messages differently.

For example, you can partition messages based on the source host:

```
worker-partition-key("${HOST}")
```

8.3 Send data to Azure Monitor and Sentinel

Starting with version 4.10.0, AxoSyslog can send data to Azure Monitor using its HTTP REST Logs ingestion API . Data sent to Azure Monitor's Log Analytics is also available from Microsoft Sentinel .

> **Note**
>
> Version 4.10 introduced the `azure-monitor-builtin()` and `azure-monitor-custom()` destinations. These were deprecated and unified as `azure-monitor()` in version 4.11.
>
> Also, the `table-name()` option of the driver has been renamed to `stream-name()` .

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `azure-monitor()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/azure/azure-monitor.conf on GitHub.

- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

- This feature requires a separate module. Install the `axosyslog-mod-cloud-auth` package on Debian/Ubuntu , or the `axosyslog-cloud-auth` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

- An Azure subscription.

- A Microsoft Entra application . You'll need the Tenant ID, App ID, and App Secret of the application to configure the AxoSyslog destination.

- A Data Collection Endpoint (DCE)
- A Data Collection Rule (DCR)
- A Log Analytics Workspace in Azure .

For details, see the Tutorial: Send data to Azure Monitor Logs with Logs ingestion API .

Configuration

The `azure-monitor()` driver sends data to the built-in tables of Azure Monitor. The body of the message (`${MESSAGE}`) must be in JSON format. The keys in the JSON array must have the same names as the columns of the table (you can use `format-json` or ‘FilterX`). If a field is empty, or Azure cannot parse it, it will be blank. The following example sends data to the syslog table .

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

destination d_azure {
  azure-monitor(
    stream-name("syslog")
    dcr-id("my-dcr-id")
    dce-uri("https://dce-uri.ingest.monitor.azure.com")
    template("$MESSAGE")
    auth(tenant-id("my-tenant-id") app-id("my-app-id") app-secret("my-app-secret"))
  );
};
```

Options

The following options are specific to the `azure-monitor()` destination. But since this destination is based on the `http()` destination, you can use the options of the `http()` destination as well if needed.

> Note: The `azure-monitor()` destination automatically configures some of these `http()` destination options as required by the Azure Monitor Logs ingestion API.

auth()

Options for OAUTH2 authentication for Azure.

To authenticate, you need to register a Microsoft Entra application . You'll need the Tenant ID, App ID, and App Secret of this application to configure the AxoSyslog destination.

app-id()

Type:	string
Default:	

Description: Application (client) ID of the Microsoft Entra application.

app-secret()

Type:	string
Default:	

Description: The Client secret of the Microsoft Entra application.

auth-url()

Type:	string
Default:	`https://login.microsoftonline.com`

Available in AxoSyslog 4.29 and later.

Description: The Microsoft Entra login host that AxoSyslog requests the OAuth2 access token from. AxoSyslog sends the token request to `<auth-url>/<tenant-id>/oauth2/v2.0/token`. Set this option to reach a national or sovereign cloud instead of the public Azure cloud. For example, for Azure Government, use `auth-url("https://login.microsoftonline.us")`.

scope()

Type:	string
Default:	`https://monitor.azure.com//.default`

Description: The scope to request for the access token.

tenant-id()

Type:	string
Default:	

Description: Directory (tenant) ID of the Microsoft Entra application.

dce-uri()

Type:	string
Default:	-

Description: The URI of your Data Collection Endpoint (DCE).

dcr-id()

Type:	string
Default:	-

Description: The ID of the Azure Monitor Data Collection Rule (DCR) where AxoSyslog sends the data.

table-name()

This option was available in version 4.10, but has been deprecated in 4.11. Use `stream-name()` instead.

stream-name()

Type:	string
Default:	-

Description: The name of the table in the Log Analytics Workspace where AxoSyslog sends the data, for example, syslog .

8.4 Send data to Google BigQuery

AxoSyslog can send data to Google Cloud BigQuery via the BigQuery Storage Write API using a high-performance gRPC-based implementation.

Prerequisites

- AxoSyslog version 4.6 or later.
- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- A Google BigQuery environment, for example, the BigQuery Sandbox .
- A BigQuery table.
- Using the Storage Write API requires one of the following OAuth scopes:
  - `https://www.googleapis.com/auth/bigquery`
  - `https://www.googleapis.com/auth/cloud-platform`
  - `https://www.googleapis.com/auth/bigquery.insertdata`

Authentication

Authentication is done via Application Default Credentials .

For authentication, the destination uses `GoogleDefaultCredentials` , which covers everything listed as ADC . In a production environment, use a service account and Workload Identity .

Configuration

To configure AxoSyslog, you'll need the name of the project, dataset, the name of the table to use, and the schema of the table.

Example configuration:

```
destination d_bigquery {
    bigquery(
        project("test-project")
        dataset("test-dataset")
        table("test-table")
        workers(8)

        schema(
            "message" => "$MESSAGE"
            "app" STRING => "$PROGRAM"
            "host" STRING => "$HOST"
            "time" DATETIME => "$ISODATE"
            "pid" INTEGER => int("$PID")
        )

        on-error("drop-property")
    );
}
```

By default, the messages are sent with one worker, one message per batch, and without compression. If you have a protobuf-formatted message , you can specify it in the `proto-var()` option, instead of using the `schema()` option.

Options

The `bigquery()` destination has the following options.

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_bigquery {
    bigquery(
      ...
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

By default, the `batch-bytes()` option of the `bigquery()` destination is 10 MB. This is an upper limit for the `bigquery()` destination. Note that due to a framework limitation, the batch might be at most 1 message larger than the set limit.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

channel-args()

Type:	arrow list
Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

compression()

Type:	boolean
Default:	no

Available in AxoSyslog version 4.5.0 and later.

Description: Enables compression in gRPC requests. Although gRPC supports various compression methods, currently only deflate is supported (which is basically the same as gzip).

dataset()

Type:	string
Default:	-

Description: The name of the dataset where AxoSyslog sends the data.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	string
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type: **number(messages)**

Default: 1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type: **yes/no**

Default: no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing,

and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()`.

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

headers()

Type:	arrow list
Default:	empty

Available in AxoSyslog 4.8 and later.

Description: Adds custom gRPC headers to each RPC call. Version 4.8 supported only static header names and values. For example:

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "axo-stream"
  )
```

Starting with version 4.9, you can use templates and macros in the header values.

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "${HOST}"
  )
```

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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-alive()

Configures how AxoSyslog sends gRPC keepalive pings .

max-pings-without-data()

Type:	integer
Default:	

Description: The maximum number of gRPC pings that can be sent when there is no data/header frame to be sent. AxoSyslog won't send any pings after this limit. Set it to 0 disable this restriction and keep sending pings.

time()

Type:	number [milliseconds]
Default:	

Description: The period (in milliseconds) after which AxoSyslog sends a gRPC keepalive ping.

timeout()

Type:	number [milliseconds]
Default:	10

Description: The time (in milliseconds) AxoSyslog waits for an acknowledgement.

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property` : Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string` : Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message` : Drop the entire message silently, without logging the error.
- `silently-drop-property` : Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string` : Convert the property to string silently, without logging the error.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

project()

Type:	string
Default:	-

Description: The ID of the Google Cloud project where AxoSyslog sends the data.

protobuf-schema()

Type:	See the description
Default:	-

Description: Sets the schema of the BigQuery table from a protobuf schema file.

```
protobuf-schema("/tmp/test.proto" => "$MESSAGE", "$PROGRAM", "$HOST", "$PID")
```

An example proto file when using the `protobuf-schema()` option:

```
syntax = "proto2";

message CustomRecord {
  optional string message = 1;
  optional string app = 2;
  optional string host = 3;
  optional int64 pid = 4;
}
```

Alternatively, you can set the schema with the `schema()` option, or use proto-var() to assign an already formatted object to the message.

proto-var()

Type:	FilterX variable
Default:	

Available in AxoSyslog 4.13 and later.

Description: Create the message contents from a FilterX variable formatted using the protobuf_message function, instead of using `schema()` or `protobuf-schema()`.

response-action()

Type:	arrow list
Default:	Depends on the driver

Available in AxoSyslog version 4.11.0 and later.

Description: Fine-tunes how AxoSyslog behaves in case of different gRPC results. You can assign specific actions to the different gRPC results, for example:

```
response-action(
  not-found => disconnect
  unavailable => drop
)
```

The following gRPC results are supported:

- `aborted`
- `already-exists`
- `cancelled`
- `data-loss`
- `deadline-exceeded`
- `failed-precondition`
- `internal`
- `invalid-argument`
- `not-found`

- `ok`
- `out-of-range`
- `permission-denied`
- `resource-exhausted`
- `unauthenticated`
- `unavailable`
- `unknown`
- `unimplemented`

The following actions are available:

- `disconnect`
- `drop`
- `retry`
- `success`

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

schema()

Type:	arrow list
Default:	-

Description: Sets the schema of the BigQuery table. On the left side of the arrow, set the name of the column and its type. On the right side, set any AxoSyslog template or macro, which gets evaluated on each log that is routed to the `bigquery()` destination.

The available column types are: `STRING`, `BYTES`, `INTEGER`, `FLOAT`, `BOOLEAN`, `TIMESTAMP`, `DATE`, `TIME`, `DATETIME`, `JSON`, `NUMERIC`, `BIGNUMERIC`, `GEOGRAPHY`, `RECORD`, `INTERVAL`. For example:

```
schema(
    "message" => "$MESSAGE"
    "app" STRING => "$PROGRAM"
    "host" STRING => "$HOST"
    "time" DATETIME => "$ISODATE"
    "pid" INTEGER => int("$PID")
)
```

Alternatively, you can set the schema with the `protobuf-schema()` option, or use proto-var() to assign an already formatted object to the message.

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.

table()

Type:	string
Default:	-

Description: The name of the Google BigQuery table where AxoSyslog sends the data.

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

url()

Type:	string
Default:	`bigquerystorage.googleapis.com`

Description: The URL of the Google BigQuery where the logs are sent.

worker-partition-autoscaling()

Type:	`yes`, `no`

Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)`, set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()`. Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()`.

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()`, `headers()`, or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()`.

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.5 ClickHouse database

Starting with version 4.9.0, AxoSyslog can send data to ClickHouse databases using its gRPC interface .

Prerequisites

- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- A self-hosted ClickHouse installation .

> **Warning**
>
> **ClickHouse Cloud doesn't support the gRPC interface currently.**

- The gRPC interface must be enabled in your ClickHouse configuration.
- To configure AxoSyslog, you'll need:
  - the name of an existing database and a table where you want to send your data, and
  - the credentials (username and password) to access the database.

Example configuration:

```
destination {
  clickhouse(
    url("localhost:9100")
    database("default")
    table("demo_table")
    user("your-username")
    password("your-password")
    schema(
      "user_id" UInt32 => $R_MSEC,
      "message" String => "$MSG",
      "timestamp" DateTime => "$R_UNIXTIME",
      "metric" Float32 => 3.14
    )
  );
};
```

If you have a protobuf-formatted message , you can specify it in the `proto-var()` option, instead of using the `schema()` option.

Options

This destination has the following options:

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_clickhouse {
    clickhouse(
      ...
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

batch-bytes()

Accepted values:	number [bytes]
Default:	4MB

Available in AxoSyslog version 4.6 and later.

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option. The batch might be at most 1 message larger than the set limit.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

OTLP has a default 4 MiB batch limit, therefore the default value for `batch-bytes()` is 4 MB, which is a bit below 4 MiB.

The batch size is calculated before compression, which is the same as the limit is calculated on the server.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number

Default:	0

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

channel-args()

Type:	arrow list

Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

compression()

Type:	boolean
Default:	`no`

Available in AxoSyslog version 4.5.0 and later.

Description: Enables compression in gRPC requests. Although gRPC supports various compression methods, currently only deflate is supported (which is basically the same as gzip).

database()

Type:	string
Default:	`default`

Description: The database where AxoSyslog sends the data.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ### Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

format()

Type:	`JSONEachRow` , `JSONCompactEachRow` , or `Protobuf`
Default:	see description

Available in AxoSyslog 4.20 and later.

Description: Specifies the data format to use when sending data to Clickhouse.

By default, `format()` is set to:

- `Protobuf` if the `proto-var()` option is set, or
- `JSONEachRow` if the `json-var()` option is set.

Starting with AxoSyslog 4.20, you can also use `format(JSONCompactEachRow)` (when `json-var()` is also set) to use a more compact, array-based JSON representation. For example:

```
destination {
  clickhouse (
    # ...
    json-var(json("$my_filterx_json_variable"))
    format("JSONCompactEachRow")
    # ...
  );
};
```

In the `JSONEachRow` format each line is a JSON object, making it more readable. For example:

```
{"id":1,"name":"foo","value":42}
{"id":2,"name":"bar","value":17}
```

In the `JSONCompactEachRow` format each row is a compact array-based row:

```
[1,"foo",42]
[2,"bar",17]
```

Note that if the data's actual format doesn't match the selected format, ClickHouse returns a `CANNOT_PARSE_INPUT_ASSERTION_FAILED` error message.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

headers()

Type:	arrow list
Default:	empty

Available in AxoSyslog 4.8 and later.

Description: Adds custom gRPC headers to each RPC call. Version 4.8 supported only static header names and values. For example:

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "axo-stream"
  )
```

Starting with version 4.9, you can use templates and macros in the header values.

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "${HOST}"
  )
```

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog created rule is there, packets can flow, otherwise the port is closed.

```
source {
    network(transport(udp)
    hook-commands(
          startup("iptables -I LOGCHAIN 1 -p udp --dport 514 -j ACCEPT")
          shutdown("iptables -D LOGCHAIN 1")
        )
     );
};
```

json-var()

Type:	string
Default:	empty string

Available in AxoSyslog 4.17 and later.

Description: The `json-var()` option accepts either a JSON template or a variable containing a JSON string, and sends it to the ClickHouse server in Protobuf/JSON mixed mode (`JSONEachRow` format). In this mode, type validation is performed by the ClickHouse server itself, so no Protobuf schema is required for communication. For example:

```
destination {
  clickhouse (
    ...
    json-var(json("{\"ingest_time\":1755248921000000000, \"body\": \"test
template\"}"))ß
    };
};
```

Using `json-var()` is mutually exclusive with the `proto-var()` , `server-side-schema()` , `schema()` , and `protobuf-schema()` options.

keep-alive()

Configures how AxoSyslog sends gRPC keepalive pings .

max-pings-without-data()

Type:	integer
Default:	

Description: The maximum number of gRPC pings that can be sent when there is no data/header frame to be sent. AxoSyslog won't send any pings after this limit. Set it to 0 disable this restriction and keep sending pings.

time()

Type:	number [milliseconds]
Default:	

Description: The period (in milliseconds) after which AxoSyslog sends a gRPC keepalive ping.

timeout()

Type:	number [milliseconds]
Default:	10

Description: The time (in milliseconds) AxoSyslog waits for an acknowledgement.

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property` : Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string` : Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message` : Drop the entire message silently, without logging the error.
- `silently-drop-property` : Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string` : Convert the property to string silently, without logging the error.

password()

Type:	string
Default:	empty string

Description: The password used for authentication.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

protobuf-schema()

Type:	See the description
Default:	-

Description: Sets the schema of the database table from a protobuf schema file.

```
protobuf-schema("/tmp/test.proto" => "$MESSAGE", "$PROGRAM", "$HOST", "$PID")
```

An example proto file when using the `protobuf-schema()` option:

```
syntax = "proto2";

message CustomRecord {
  optional string message = 1;
  optional string app = 2;
  optional string host = 3;
  optional int64 pid = 4;
}
```

Alternatively, you can set the schema with the `schema()` option, use proto-var() to assign an already formatted object to the message, or use a JSON template with the json-var() option.

proto-var()

Type:	FilterX variable
Default:	

Available in AxoSyslog 4.13 and later.

Description: Create the message contents from a FilterX variable formatted using the protobuf_message function, instead of using `schema()` or `protobuf-schema()`.

response-action()

Type:	arrow list
Default:	Depends on the driver

Available in AxoSyslog version 4.11.0 and later.

Description: Fine-tunes how AxoSyslog behaves in case of different gRPC results. You can assign specific actions to the different gRPC results, for example:

```
response-action(
  not-found => disconnect
  unavailable => drop
)
```

The following gRPC results are supported:

- `aborted`
- `already-exists`
- `cancelled`
- `data-loss`
- `deadline-exceeded`
- `failed-precondition`
- `internal`
- `invalid-argument`
- `not-found`

- `ok`
- `out-of-range`
- `permission-denied`
- `resource-exhausted`
- `unauthenticated`
- `unavailable`
- `unknown`
- `unimplemented`

The following actions are available:

- `disconnect`
- `drop`
- `retry`
- `success`

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

schema()

Type:	arrow list
Default:	

Description: Sets the schema of the database table. On the left side of the arrow, set the name of the column and its type. On the right side, set any AxoSyslog template or macro, which gets evaluated on each log that is routed to the destination. For example:

```
schema(
  "user_id" UInt32 => $R_MSEC,
  "message" String => "$MSG",
  "timestamp" DateTime => "$R_UNIXTIME",
  "metric" Float32 => 3.14
)
```

Alternatively, you can set the schema with the `protobuf-schema()` option, use proto-var() to assign an already formatted object to the message, or use a JSON template with the json-var() option.

You can find the available column types in the official ClickHouse documentation .

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.

server-side-schema()

Type:	string
Default:	-

By default, sending data to ClickHouse doesn't propagate the type information of the data fields. The `server-side-schema()` option provides a solution for that using the ClickHouse format schema . Using a server-side schema is needed when you're using

complex types, like `DateTime` or `LowCardinality` .

1. Create a `.proto` file that matches the `schema()` of your AxoSyslog configuration. For details on formatting, see the ClickHouse documentation . For example:

```
syntax = "proto3";

message MessageType {
  uint32 user_id = 3;
  string message = 1;
  string timestamp = 2;
  float metric = 4;
};
```

2. Copy the `.proto` file to your ClickHouse server, into the directory set in the `proto_schema_path` option of your ClickHouse configuration.

3. Reference that schema in the `clickhouse()` destination of your AxoSyslog configuration. The parameter of `server-side-schema()` is `<name-of-the-proto-file>:<identifier-after-message-in-the-proto-file>` . So if in the previous step you named the proto file `my-proto-file.proto` with the sample content, the parameter will be `my-proto-file:MessageType` . For example:

```
destination {
  clickhouse(
    database("default")
    table("demo_table")
    user("your-username")
    password("your-password")
    schema(
      "user_id" UInt32 => $R_MSEC,
      "message" String => "$MSG",
      "timestamp" DateTime => "$R_UNIXTIME",
      "metric" Float32 => 3.14
    )
    server-side-schema("<my_proto_file_on_server>:<my_message_schema_name>")
  );
};
```

table()

Type:	string
Default:	-

Description: The name of the table where AxoSyslog sends the data.

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

url()

Type:	string

Default:	`localhost:9100`

Description: The address of the gRPC receiver in a `host:port` format. If the port is omitted, it will default to `443`. Examples: `127.0.0.1:9100`, `[::1]:1234`, `example.com` (equivalent to `example.com:443`).

user()

Type:	string
Default:	empty string

Description: The username used for authentication.

worker-partition-autoscaling()

Type:	`yes`, `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)`, set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()`. Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()`.

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()`, `headers()`, or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()`.

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

Warning

`Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.6 collectd: Send metrics to collectd

The `collectd()` destination uses the unixsock plugin of the collectd application to send log messages to the collectd system statistics collection daemon . You must install and configure collectd separately before using this destination.

Prerequisites

- AxoSyslog version 3.20 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `collectd()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/collectd/plugin.conf on GitHub.

Declaration:

```
destination d_collectd {
   collectd(
     socket("<path-to-collectd-socket>"),
     host("${HOST}"),
     plugin("${PROGRAM}"),
     type("<type-of-the-collected-metric>"),
     values("<metric-sent-to-collectd>"),
   );
 };
```

Example: Using the collectd() driver

The following example uses the name of the application sending the log message as the plugin name, and the value of the ${SEQNUM} macro as the value of the metric sent to collectd.

```
destination d_collectd {
   collectd(
     socket("/var/run/collectd-unixsock"),
     host("${HOST}"),
     plugin("${PROGRAM}"),
     type("gauge"),
     type_instance("seqnum"),
     values("${SEQNUM}"),
   );
 };
```

8.6.1 collectd() destination options

The `collectd()` destination has the following options. The `plugin()` and `type()` options are required options. You can also set other options of the underlying `unix-stream()` driver (for example, socket buffer size).

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> ### Note
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> ### Warning
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> ### Note
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ## Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

flush-lines()

Type:	number
Default:	Use global setting (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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:	string, macro, or template
Default:	${HOST}

Description: The hostname that is passed to collectd. By default, AxoSyslog uses the host from the log message as the hostname.

```
    type("gauge"),
```

interval()

Type:	integer
Default:	60

Description: The interval in which the data is collected.

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 connections, server-side (source) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the source.

plugin()

Type:	string
Default:	

Description: The name of the plugin that submits the data to collectd. For example:

```
plugin("${PROGRAM}"),
```

plugin-instance()

Type:	string
Default:	

Description: The name of the plugin-instance that submits the data to collectd.

socket()

Type:	path
Default:	/var/run/collectd-unixsock

Description: The path to the socket of collectd. For details, see the collectd-unixsock(5) manual page .

```
type("gauge"),
```

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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), AxoSyslog 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 AxoSyslog waits for identical messages.

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.

type()

Type:	string or template
Default:	

Description: Identifies the type and number of values passed to collectd. For details, see the types.db manual page . For example:

```
    type("gauge"),
```

type-instance()

Type:	string
Default:	

Description: For example:

```
    type-instance("seqnum"),
```

values()

Type:	string, macro,or template
Default:	U

Description: Colon-separated list of the values to send to collectd. For example:

```
    values("${SEQNUM}"),
```

8.7 discord: Send alerts and notifications to Discord

The `discord()` destination driver sends messages to Discord using Discord Webhook . For the list of available optional parameters, see Discord destination options.

Prerequisites

- AxoSyslog version 3.33 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `discord()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/discord/discord.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- To send messages to Discord, you must setup webhooks. For details, see: Discord: Intro to Webhooks .

Declaration:

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

destination {
    discord(url("https://discord.com/api/webhooks/x/y"));
};
```

By default the message sending is throttled to 5 message/sec, see Discord: Rate Limits . To change this, use the `throttle()` option.

Example: Using the discord() driver

The following example sends messages with custom avatar, and text-to-speech enabled.

```
@include "scl.conf"
destination d_discord {
    discord(
        url("https://discord.com/api/webhooks/x/y")
        avatar-url("https://example.domain/any_image.png")
        username("$HOST-bot") # Custom bot name, accepts macros
        tts(true) # Text-to-Speech message
        template("${MSG:-[empty message]}") # Message to send, can't be empty
    );
}
```

8.7.1 Discord destination options

The `discord()` destination of AxoSyslog can directly post log messages to web services using the HTTP protocol. The `discord()` destination has the following options.

avatar-url()

Type:	URL
Default:	N/A

Description: A hyperlink for icon of the author to be displayed in Discord. For details, see the `avatar_url` option in the Discord documentation .

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cert-file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate (or a certificate chain) in PEM format, suitable as a TLS certificate, matching the private key set in the `key-file()` option. The AxoSyslog application uses this certificate to authenticate the AxoSyslog client on the destination server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that AxoSyslog 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 AxoSyslog. 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 AxoSyslog receives TLS-encrypted connections, the order of ciphers set on the AxoSyslog server takes precedence over the client settings.

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

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

Declaration:

```
   destination d_http {
        http(
            url("http://127.0.0.1:8080")
            tls(
                ca-dir("dir")
                ca-file("ca")
                cert-file("cert")
                cipher-suite("cipher")
                key-file("key")
                peer-verify(yes|no)
                ssl-version(<the permitted SSL/TLS version>)
            )
        );
    };
```

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

The `http()` destination supports only unencrypted key files (that is, the private key cannot be password-protected).

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()` , `ca-file()` , `cert-file()` , `cipher-suite()` , `key-file()` , `peer-verify()` , and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()` .

Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

max-msg-length()

Type:	Number
Default:	2000

Description: Removes every character above the set limit. For details, see the content option in the Discord documentation .

peer-verify()

Accepted values:	`yes` or `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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()` , `ca-file()` , `cert-file()` , `cipher-suite()` , `key-file()` , `peer-verify()` , and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()` . Avoid mixing the two methods. 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:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

Alternatively, you can leave the HTTP as-is, in which case the driver leaves the default `http_proxy` and `https_proxy` environment variables unmodified.

> **Note**
>
> **Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.**

Example: the proxy() option in configuration

The following example illustrates including the `proxy()` option in your configuration.

```
destination {
   http( url("SYSLOG_SERVER_IP:PORT") proxy("PROXY_IP:PORT") method("POST"));
 };
```

response-action()

Type:	list
Default:	N/A (see below)

Description: Specifies what AxoSyslog does with the log message, based on the response code received from the HTTP server. If the server returns a status code beginning with 2 (for example, 200), AxoSyslog assumes the message was successfully sent. Otherwise, the action listed in the following table is applied. For status codes not listed in the following table, if the status code begins with 2 (for example, 299), AxoSyslog assumes the message was successfully sent. For other status codes, AxoSyslog disconnects. The following actions are possible:

- `disconnect` : Keep trying to resend the message indefinitely.
- `drop` : Drop the message without trying to resend it.
- `retry` : Retry sending the message for a maximum of `retries()` times (3 by default).
- `success` : Assume the message was successfully sent.

```
|------+-----------------------------------+------------|
 | code | explanation                       | action     |
 |------+-----------------------------------+------------|
 |  100 | "Continue"                        | disconnect |
 |  101 | "Switching Protocols"             | disconnect |
 |  102 | "Processing"                      | retry      |
 |  103 | "Early Hints"                     | retry      |
 |  200 | "OK"                              | success    |
 |  201 | "Created"                         | success    |
 |  202 | "Accepted"                        | success    |
 |  203 | "Non-Authoritative Information"   | success    |
 |  204 | "No Content"                      | success    |
 |  205 | "Reset Content"                   | success    |
 |  206 | "Partial Content"                 | success    |
 |  300 | "Multiple Choices"                | disconnect |
 |  301 | "Moved Permanently"               | disconnect |
 |  302 | "Found"                           | disconnect |
 |  303 | "See Other"                       | disconnect |
 |  304 | "Not Modified"                    | retry      |
 |  307 | "Temporary Redirect"              | disconnect |
 |  308 | "Permanent Redirect"              | disconnect |
```

```
|  400 | "Bad Request"                       | disconnect |
|  401 | "Unauthorized"                      | disconnect |
|  402 | "Payment Required"                  | disconnect |
|  403 | "Forbidden"                         | disconnect |
|  404 | "Not Found"                         | disconnect |
|  405 | "Method Not Allowed"                | disconnect |
|  406 | "Not Acceptable"                    | disconnect |
|  407 | "Proxy Authentication Required"     | disconnect |
|  408 | "Request Timeout"                   | disconnect |
|  409 | "Conflict"                          | disconnect |
|  410 | "Gone"                              | drop       |
|  411 | "Length Required"                   | disconnect |
|  412 | "Precondition Failed"               | disconnect |
|  413 | "Payload Too Large"                 | disconnect |
|  414 | "URI Too Long"                      | disconnect |
|  415 | "Unsupported Media Type"            | disconnect |
|  416 | "Range Not Satisfiable"             | drop       |
|  417 | "Expectation Failed"                | disconnect |
|  418 | "I'm a teapot"                      | disconnect |
|  421 | "Misdirected Request"               | disconnect |
|  422 | "Unprocessable Entity"              | drop       |
|  423 | "Locked"                            | disconnect |
|  424 | "Failed Dependency"                 | drop       |
|  425 | "Too Early"                         | drop       |
|  426 | "Upgrade Required"                  | disconnect |
|  428 | "Precondition Required"             | retry      |
|  429 | "Too Many Requests"                 | disconnect |
|  431 | "Request Header Fields Too Large"   | disconnect |
|  451 | "Unavailable For Legal Reasons"     | drop       |
|  500 | "Internal Server Error"             | disconnect |
|  501 | "Not Implemented"                   | disconnect |
|  502 | "Bad Gateway"                       | disconnect |
|  503 | "Service Unavailable"               | disconnect |
|  504 | "Gateway Timeout"                   | retry      |
|  505 | "HTTP Version Not Supported"        | disconnect |
|  506 | "Variant Also Negotiates"           | disconnect |
|  507 | "Insufficient Storage"              | disconnect |
|  508 | "Loop Detected"                     | drop       |
|  510 | "Not Extended"                      | disconnect |
|  511 | "Network Authentication Required"   | disconnect |
|------+-------------------------------------+------------|
```

To customize the action to take for a particular response code, use the following format: `response-action(<response-code> => <action>` . To customize multiple response code-action pairs, separate them with a comma, for example:

```
 http(
    url("http://localhost:8080")
    response-action(418 => drop, 404 => retry)
);
```

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()` , AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

To handle HTTP error responses, if the HTTP server returns 5xx codes, AxoSyslog will attempt to resend messages until the number of attempts reaches `retries` . If the HTTP server returns 4xx codes, AxoSyslog will drop the 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 AxoSyslog. 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.

> **Warning**
>
> **Hazard of data loss! Make sure to include a fallback value, as if the template gets resolved to an empty string, Discord rejects the message.**

throttle()

Type:	number
Default:	5

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.

For more information, see Discord: Rate Limits .

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.

tts()

Type:	`true` or `false`
Default:	false

Description: Enables TTS (Text-To-Speech) mode. For more information, see the tts option in the Discord documentation .

url()

Type:	URL
Default:	N/A

Description: The webhook URL of the Discord server/channel. For more information, see Discord: Intro to Webhooks .

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.

username()

Type:	string
Default:	N/A

Description: Overrides the default username of the webhook. For details, see the username option in the Discord documentation .

use-system-cert-store()

Type:	`yes` or `no`
Default:	`no`

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation .

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.8 elasticsearch2: DEPRECATED - Send messages directly to Elasticsearch version 2.0 or higher

> **Warning**
>
> **This destination is deprecated and will be removed from a future version of AxoSyslog. We recommend using the elasticsearch-http: Send messages to Elasticsearch HTTP Bulk API destination instead.**

Starting with version 3.7 of AxoSyslog 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 AxoSyslog `elasticsearch2` destination:

- Since AxoSyslog uses Java libraries, the `elasticsearch2` destination has significant memory usage.

Prerequisites

The Java implementation of this destination is no longer shipped with AxoSyslog: the `elasticsearch2` Java module isn't part of the `axosyslog-mod-java` (Debian/Ubuntu) or `axosyslog-java` (RHEL and compatible distributions) package. Use the elasticsearch-http: Send messages to Elasticsearch HTTP Bulk API destination instead.

Declaration:

```
@include "scl.conf"

 elasticsearch2(
     index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
     type("test")
     cluster("syslog-ng")
 );
```

Example: Sending log data to Elasticsearch version 2.x and above

The following example defines an `elasticsearch2` destination that sends messages in transport mode to an Elasticsearch server running on the localhost, using only the required parameters.

```
@include "scl.conf"

 destination d_elastic {
     elasticsearch2(
         index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
         type("test")
     );
 };
```

The following example sends 10000 messages in a batch, in transport mode, and includes a custom unique ID for each message.

```
@include "scl.conf"

 options {
     threaded(yes);
     use-uniqid(yes);
 };

 source s_syslog {
     syslog();
 };

 destination d_elastic {
     elasticsearch2(
         index("syslog-ng_${YEAR}.${MONTH}.${DAY}")
         type("test")
         cluster("syslog-ng")
         client-mode("transport")
         custom-id("${UNIQID}")
         flush-limit("10000")
     );
 };

 log {
     source(s_syslog);
     destination(d_elastic);
     flags(flow-control);
 };
```

Example: Sending log data to Elasticsearch using the HTTP REST API

The following example send messages to Elasticsearch over HTTP using its REST API:

```
@include "scl.conf"

source s_network {
    network(port(5555));
};

destination d_elastic {
    elasticsearch2(
        client-mode("http")
        cluster("es-syslog-ng")
        index("x201")
        cluster-url("http://192.168.33.10:9200")
        type("slng_test_type")
        flush-limit("0")
    );
};

log {
    source(s_network);
    destination(d_elastic);
    flags(flow-control);
};
```

Verify the certificate of the Elasticsearch server and perform certificate authentication (this is actually a mutual, certificate-based authentication between the AxoSyslog 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 AxoSyslog interacts with Elasticsearch.
- For the list of options, see Elasticsearch2 destination options (DEPRECATED).

> **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.**

8.8.1 Prerequisites

To send messages from AxoSyslog to Elasticsearch, complete the following steps.

Steps:

1. Download and install the Java Runtime Environment (JRE), 2.x (or newer). The AxoSyslog `elasticsearch2` destination is tested and supported when using the Oracle implementation of Java. Other implementations are untested and unsupported, they may or may not work as expected.

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

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

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

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

8.8.2 How AxoSyslog interacts with Elasticsearch

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

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

      This method ensures reliable message transfer, but is slow (about 1000 messages/second).
    - If `flush-limit` is higher than 1: AxoSyslog sends messages in a batch, and receives the response asynchronously. In case of a problem, AxoSyslog 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 AxoSyslog recognizes the error. In node mode, about 1000 messages can be lost.
    - If `concurrent-requests` is higher than 1, AxoSyslog can send multiple batches simultaneously, increasing performance (and also the number of messages that can be lost in case of an error).
- Version 3.10 and newer of AxoSyslog automatically converts the timestamp (date) of the message to UTC, as needed by Elasticsearch and Kibana.

8.8.3 Client modes

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

HTTP mode

The AxoSyslog application sends messages over HTTP using the REST API of Elasticsearch, and uses the `cluster-url()` and `cluster()` options from the AxoSyslog configuration file. In HTTP mode, AxoSyslog `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTP mode is available in AxoSyslog 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 AxoSyslog application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), AxoSyslog does not have any direct influence on it.

HTTPS mode

The AxoSyslog 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 AxoSyslog configuration file. In HTTPS mode, AxoSyslog `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTPS mode is available in AxoSyslog 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 AxoSyslog application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), AxoSyslog does not have any direct influence on it.

Transport mode

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

Node mode

The AxoSyslog 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 AxoSyslog to Elasticsearch 2.x. For Elasticsearch versions 5.x and newer, use HTTPS mode. For details on configuring Search Guard mode, see Search Guard.

8.8.4 Search Guard

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 AxoSyslog to Elasticsearch 2 and newer. To configure AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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
```

8.8.5 Elasticsearch2 destination options (DEPRECATED)

> **Warning**
>
> **This destination is deprecated and will be removed from a future version of AxoSyslog. We recommend using the elasticsearch-http: Send messages to Elasticsearch HTTP Bulk API destination instead.**

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

Required options:

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

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

client-lib-dir()

Type:	string
Default:	The AxoSyslog 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 AxoSyslog configuration (for example, because you have multiple Java-based destinations), AxoSyslog 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 AxoSyslog application sends messages over HTTP using the REST API of Elasticsearch, and uses the `cluster-url()` and `cluster()` options from the AxoSyslog configuration file. In HTTP mode, AxoSyslog `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTP mode is available in AxoSyslog 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 AxoSyslog application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), AxoSyslog does not have any direct influence on it.

HTTPS mode

The AxoSyslog 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 AxoSyslog configuration file. In HTTPS mode, AxoSyslog `elasticsearch2` driver can send log messages to every Elasticsearch version, including 1.x-6.x. Note that HTTPS mode is available in AxoSyslog 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 AxoSyslog application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), AxoSyslog does not have any direct influence on it.

Transport mode

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

Node mode

The AxoSyslog 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 AxoSyslog to Elasticsearch 2.x. For Elasticsearch versions 5.x and newer, use HTTPS mode. For details on configuring Search Guard mode, see Search Guard.

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")")` .

In version 3.10 and newer, you can list multiple servers in HTTP and HTTPS mode in the `cluster-url()` and `server()` options. The AxoSyslog application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), AxoSyslog 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 AxoSyslog 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() .

> **Warning**
>
> **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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()` .

compaction()

Type:	**yes/no**

Default: no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type: string

Default: N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

Warning

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.

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

Warning

Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

flush-limit()

Type:	number
Default:	5000

Description: The number of messages that AxoSyslog sends to the Elasticsearch server in a single batch.

- If `flush-limit` is set to 1: AxoSyslog sends the message reliably: it sends a message to Elasticsearch, then waits for a reply from Elasticsearch. In case of failure, AxoSyslog repeats sending the message, as set in the `retries()` parameter. If sending the message fails for `retries()` times, AxoSyslog drops the message.

  This method ensures reliable message transfer, but is slow (about 1000 messages/second).

- If `flush-limit` is higher than 1: AxoSyslog sends messages in a batch, and receives the response asynchronously. In case of a problem, AxoSyslog 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 AxoSyslog recognizes the error. In node mode, about 1000 messages can be lost.

- If `concurrent-requests` is higher than 1, AxoSyslog can send multiple batches simultaneously, increasing performance (and also the number of messages that can be lost in case of an error).

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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, AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 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.conf` configuration file.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

on-error()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.

- `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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()` , AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

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 AxoSyslog application will use these destination servers in load-balancing fashion. Note that load-balancing is handled by an external library (Jest), AxoSyslog 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` or `no`
Default:	no

Description: By default, when connecting to an Elasticsearch cluster, AxoSyslog checks the state of the cluster. If the primary shards of the cluster are not active, AxoSyslog 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:	`$(format-json --scope rfc5424 --exclude DATE --key ISODATE @timestamp=${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 AxoSyslog 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")`.

8.9 Send messages to Elasticsearch data streams

AxoSyslog can send messages and metrics to Elasticsearch data streams to store your log and metrics data as time series data.

Prerequisites

- AxoSyslog version 4.8 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `elasticsearch-datastream()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/elasticsearch/elastic-datastream.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- Username and password for an account that can send data to Elasticsearch data streams.

Configuration

Minimal configuration:

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

destination d_elastic_data_stream {
  elasticsearch-datastream(
    url("https://elastic-endpoint:9200/my-data-stream/_bulk")
    user("elastic")
    password("ba253DOn434Tc0pY22OI")
  );
};
```

Options

Usually you just set the `url()`, `user()`, and `password()` options.

Since this destination is based on the `http()` destination, you can use the options of the `http()` destination if needed.

8.10 elasticsearch-http: Send messages to Elasticsearch HTTP Bulk API

Version 3.21 of AxoSyslog can directly post log messages to an Elasticsearch deployment using the Elasticsearch Bulk API over the HTTP and Secure HTTP (HTTPS) protocols.

HTTPS connection, as well as password- and certificate-based authentication is supported. The content of the events is sent in JSON format.

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `elasticsearch-http()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/elasticsearch/elastic-http.conf on GitHub.

- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

```
d_elasticsearch_http {
     elasticsearch-http(
         index("<elasticsearch-index-to-store-messages>")
         url("https://your-elasticsearch-server1:9200/_bulk")
         type("<type-of-the-index>")
     );
 };
```

Use an empty string to omit the type from the index: `type("")`. For example, you need to do that when using Elasticsearch 7 or newer, and you use a mapping in Elasticsearch to modify the type of the data.

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

Alternatively, you can leave the HTTP as-is, in which case the driver leaves the default `http_proxy` and `https_proxy` environment variables unmodified.

For more detailed information about these environment variables, see the libcurl documentation .

> **Note**
>
> **Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.**

Example: Sending log data to Elasticsearch

The following example defines an `elasticsearch-http()` destination, with only the required options.

```
destination d_elasticsearch_http {
    elasticsearch-http(
        index("<name-of-the-index>")
        type("<type-of-the-index>")
        url("http://my-elastic-server:9200/_bulk")
    );
};


log {
    source(s_file);
    destination(d_elasticsearch_http);
    flags(flow-control);
};
```

The following example uses mutually-authenticated HTTPS connection, templated index, and also sets the `type()` and some other options.

```
destination d_elasticsearch_https {
     elasticsearch-http(
         url("https://node01.example.com:9200/_bulk")
         index("test-${YEAR}${MONTH}${DAY}")
         time-zone("UTC")
         type("test")
         workers(4)
         batch-lines(16)
         timeout(10)
         tls(
             ca-file("ca.pem")
             cert-file("syslog_ng.crt.pem")
             key-file("syslog_ng.key.pem")
             peer-verify(yes)
         )
     );
 };
```

8.10.1 Batch mode and load balancing

The `elasticsearch-http()` destination automatically sends multiple log messages in a single HTTP request, increasing the rate of messages that your Elasticsearch deployment can consume. For details on adjusting and fine-tuning the batch mode of the `elasticsearch-http()` destination, see the following section.

Batch size

The `batch-bytes()`, `batch-lines()`, `batch-timeout()`, and `batch-idle-timeout()` options of the destination determine how many log messages AxoSyslog sends in a batch. The `batch-lines()` option determines the maximum number of messages AxoSyslog puts in a batch in. This can be limited based on size and time:

- AxoSyslog sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. You can also set `batch-idle-timeout()` to close the batch if too much time elapses since the last message. That way the destination receives every message in a timely manner even if there are no more messages.
- AxoSyslog sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Example: HTTP batch mode

In the following example, a batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         index("<elasticsearch-index-to-store-messages>")
         type("")
         batch-lines(100)
         batch-bytes(512Kb)
         batch-timeout(10000)
     );
 };
```

Load balancing between multiple Elasticsearch indexers

Starting with version 3.19, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, AxoSyslog sends log messages to the specified URLs in a load-balance fashion. This means that AxoSyslog sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order AxoSyslog has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

> **Warning**
>
> **If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.**

Starting with version AxoSyslog version 3.22, you can use any of the following formats to specify multiple URLs:

```
url("server1", "server2", "server3"); # comma-separated strings
url("server1" "server2" "server3"); # space-separated strings
url("server1 server2 server3"); # space-separated within a single string
```

Example: HTTP load balancing

The following destination sends log messages to 3 different Elasticsearch indexer nodes. Each node is assigned a separate worker thread. A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
   destination d_elasticsearch-http {
        elasticsearch-http(url("http://your-elasticsearch-server1:9200/_bulk"
"http://your-elasticsearch-server2:9200/_bulk" "http://your-elasticsearch-
server3:9200/_bulk")
            batch-lines(100)
            batch-bytes(512Kb)
            batch-timeout(20000)
            persist-name("d_elasticsearch-http-load-balance")
        );
    };
```

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.10.2 elasticsearch-http() destination options

The `elasticsearch-http` destination of AxoSyslog can directly post log messages to an Elasticsearch deployment using the Elasticsearch Bulk API over the HTTP and Secure HTTP (HTTPS) protocols. The `elasticsearch-http` destination has the following options. The required options are: `index()`, `type()`, and `url()`.

This destination is available in AxoSyslog version 3.21 and later.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

For details on how this option influences batch mode, see Batch mode and load balancing

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	25

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

For details on how this option influences batch mode, see Batch mode and load balancing

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

For details on how this option influences batch mode, see Batch mode and load balancing

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The AxoSyslog application uses this certificate to validate the certificate of the HTTPS server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cert-file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate (or a certificate chain) in PEM format, suitable as a TLS certificate, matching the private key set in the `key-file()` option. The AxoSyslog application uses this certificate to authenticate the AxoSyslog client on the destination server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()` , `ca-file()` , `cert-file()` , `cipher-suite()` , `key-file()` , `peer-verify()` , and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()` .

Avoid mixing the two methods. In case you do specify TLS options in both ways, the one that comes later in the configuration file will take effect.

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that AxoSyslog 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 AxoSyslog. 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 AxoSyslog receives TLS-encrypted connections, the order of ciphers set on the AxoSyslog server takes precedence over the client settings.

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

custom-id()

Accepted values:	string
Default:	empty string

Description: Sets the specified value as the ID of the Elasticsearch index (`_id`).

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, AxoSyslog separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option.

For details on how this option influences batch mode, see Batch mode and load balancing

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value

pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ## Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog created rule is there, packets can flow, otherwise the port is closed.

```
source {
    network(transport(udp)
    hook-commands(
          startup("iptables -I LOGCHAIN 1 -p udp --dport 514 -j ACCEPT")
          shutdown("iptables -D LOGCHAIN 1")
        )
     );
};
```

index()

Accepted values:	string or template
Default:	None

Description: The name of the Elasticsearch index where Elasticsearch will store the messages received from AxoSyslog. This option is mandatory for this destination.

You can use macros and template functions, but you must ensure that the resolved template contains only characters that Elasticsearch permits in the name of the index. The AxoSyslog application does not validate the name of the index. For details on the characters permitted in the name of Elasticsearch indices, see the documentation of Elasticsearch.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

key-file()

Accepted values:	Filename

Default: none

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

This destination supports only unencrypted key files (that is, the private key cannot be password-protected).

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

password()

Type:	string
Default:	

Description: The password that AxoSyslog uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	`yes` or `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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

proxy()

Type:	
Default:	empty

Description:

Format in configuration:

```
destination {
   http( url("SYSLOG_SERVER_IP:PORT") proxy("PROXY_IP:PORT") method("POST"));
 };
```

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

To handle HTTP error responses, if the HTTP server returns 5xx codes, AxoSyslog will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, AxoSyslog will drop the messages.

ssl-version()

Type:	string

Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2` , `sslv3` , `tlsv1` , `tlsv1_0` , `tlsv1_1` , `tlsv1_2` , `tlsv1_3` .

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

type()

Type:	string or template
Default:	N/A

Description: The type of the Elasticsearch index.

Use an empty string to omit the type from the index: `type("")`. For example, you need to do that when using Elasticsearch 7 or newer, and you use a mapping in Elasticsearch to modify the type of the data.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

timeout()

Type:	number [seconds]
Default:	10

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded.

url()

Type:	URL or list of URLs, for example, url(“site1” “site2”)

Default: N/A

Description: Specifies the hostname or IP address and optionally the port number of the Elasticsearch indexer. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://your-elasticsearch-indexer.server:8088/_bulk`

This option is mandatory for this destination.

Make sure that the URL ends with `_bulk` , this is the Elasticsearch API endpoint that properly parses the messages sent by AxoSyslog.

In case the server on the specified URL returns a redirect request, AxoSyslog automatically follows maximum 3 redirects. Only HTTP and HTTPS based redirections are supported.

Starting with version 3.19, you can specify multiple URLs, for example, `url("site1" "site2")` . In this case, AxoSyslog sends log messages to the specified URLs in a load-balance fashion. This means that AxoSyslog sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order AxoSyslog has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

Warning

If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.

Starting with version AxoSyslog version 3.22, you can use any of the following formats to specify multiple URLs:

```
url("server1", "server2", "server3"); # comma-separated strings
url("server1" "server2" "server3"); # space-separated strings
url("server1 server2 server3"); # space-separated within a single string
```

user()

Type: string

Default:	

Description: The username that AxoSyslog uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	`yes` or `no`
Default:	`no`

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation .

workers()

Type:	integer
Default:	4

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.11 file: Store messages in plain-text files

The file driver is one of the most important destination drivers. 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, AxoSyslog can store the messages of client hosts in a separate file for each host. For more information on available macros see Macros of AxoSyslog.

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

Warning

Since the state of each created file must be tracked by `syslog-ng`, it consumes some memory for each file. If no new messages are written to a file within 60 seconds (controlled by the `time-reap()` global option), it is closed, and its state is freed.

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

8.11.1 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:

> **Warning**
>
> **When creating several thousands separate log files, AxoSyslog 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 AxoSyslog or the global `ulimit` parameter of your host. For setting the `--fd-limit` command-line parameter ofAxoSyslog see the The syslog-ng manual page. For setting the `ulimit` parameter of the host, see the documentation of your operating system.**

create-dirs()

Type:	yes or no
Default:	no

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

dir-group()

Type:	string
Default:	Use the global settings

Description: The group of the directories created by `syslog-ng`. To preserve the original properties of an existing directory, use the option without specifying an attribute: `dir-group()`.

dir-owner()

Type:	string

Default:	Use the global settings

Description: The owner of the directories created by `syslog-ng`. To preserve the original properties of an existing directory, use the option without specifying an attribute: `dir-owner()`.

Starting with version 3.16, the default value of this option is -1, so AxoSyslog 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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

Warning

Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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.

> **Note**
>
> **The `file` destination uses multiple threads only if the destination filename contains macros.**

flush-lines()

Type:	number
Default:	Use global setting (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

fsync()

Type:	yes or no
Default:	no

Description: Forces an `fsync()` call on the destination fd after each write.

> **Note**
>
> **Enabling this option may seriously degrade performance.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 3.4 and later.

overwrite-if-older()

Type:	number (seconds)
Default:	0

Description: If set to a value higher than 0, AxoSyslog 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 AxoSyslog 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, AxoSyslog 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).

> **Warning**
>
> **Hazard of data loss! If the size of the incoming message is larger than the previously set `pad-size()` value, AxoSyslog 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), AxoSyslog 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 AxoSyslog waits for identical messages.

symlink-as()

Type:	Filename
Default:	N/A

Description: The configured file name will be used as a symbolic link to the last created file by file destination.

Example: symlink-as()

An example when time-based macro is used:

```
file("/var/log/cron.${YEAR}${MONTH}" symlink-as("/var/log/cron"));
```

In this case the `/var/log/cron` should point to the current month.

> **Note**
>
> **The symlink uses the same permissions as the file destination.**

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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

time-reap()

Accepted values:	number (seconds)
Default:	60 or 0, see description for details

Description: The time to wait in seconds before an idle destination file or pipe is closed. Note that only destination files having macros in their filenames are closed automatically.

Starting with version 3.23, the way how `time-reap()` works is the following.

1. If the `time-reap()` option of the destination is set, that value is used, for example:

```
destination d_fifo {
    pipe(
        "/tmp/test.fifo",
        time-reap(30)  # sets time-reap() for this destination only
    );
};
```

2. If the `time-reap()` option of the destination is not set, and the destination does not use a template or macro in its filename or path, `time-reap()` is automatically set to 0. For example:

```
destination d_fifo {
    pipe(
        "/tmp/test.fifo",
    );
};
```

3. Otherwise, the value of the global `time-reap()` option is used, which defaults to 60 seconds.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

8.12 Google Pub/Sub gRPC

Starting with version 4.10.0, AxoSyslog can send data to Google Cloud Pub/Sub using its gRPC interface .

Prerequisites

- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- - A Google Pub/Sub subscription .
  - An IAM service account that AxoSyslog uses for authentication.
  - A Google Cloud project that has the Pub/Sub API enabled.

  For details, see the Google Pub/Sub tutorial .

Configuration

To configure AxoSyslog, you'll need the name of the project and the topic where you want to send your data.

Example configuration:

```
destination d_pubsub_grpc {
  google-pubsub-grpc(
    project("my_project")
    topic($topic)

    data($MESSAGE)
    attributes(
      timestamp => $S_ISODATE,
      host => $HOST,
    )

    workers(4)
    batch-timeout(1000) # ms
    batch-lines(1000)
  );
};
```

You can change the default endpoint using `service_endpoint()` .

Options

The following options are specific to the `google-pubsub-grpc()` destination.

attributes()

Type:	string
Default:	`"--scope rfc5424,all-nv-pairs --exclude MESSAGE"`

Description: A JSON object representing key-value pairs for the Pub/Sub Event, formatted as AxoSyslog value-pairs . By default, the destination sends the RFC5424 fields as attributes. If you want to send different fields, override the default template. By default, the message part is sent in the `data()` option.

If you want to dynamically create the `data()` and `attributes()` part of the message, use the `protovar()` attribute.

auth()

Options for cloud-related authentication. Currently only the GCP Service Account authentication is supported.

Specify the JSON file storing the key to the service account like this:

```
auth(
    service-account(
      key("/path/to/service-account-key.json")
    )
  )
```

service-account()

Authenticate to a service account using Service Account Key-Based Authentication. This method works both inside and outside GCP It uses a service account key generated and downloaded through the GCP IAM & Admin console. The long-term service account key is used to generate short-term tokens for authentication (also called self-signed JWT).

audience()

Type:	string
Default:	

> **Note**
>
> - When using the `google-pubsub()` destination, the `audience()` option is set to `https://pubsub.googleapis.com/google.pubsub.v1.Publisher`. Don't change it.
> - When using the `google-pubsub()-grpc` destination, you don't need to set the `audience()` option because the underlying gRPC API recognizes it automatically.
> - You cannot use `audience()` and `scope()` together.

key()

Type:	string (path)
Default:	

Path to the service account key.

scope()

Type:	string
Default:	

Available in AxoSyslog version 4.15 and later.

URL for a scope claim for authentication instead of audience . For example:

```
...
cloud-auth(
  gcp(
    service-account(
      key("/path/to/secret.json")
      scope("https://www.googleapis.com/auth/example-scope")
    )
  )
)
```

> **Note**
>
> **You cannot use `audience()` and `scope()` together.**

token-validity-duration()

Type:	integer (seconds)
Default:	`3600`

user-managed-service-account()

Available in AxoSyslog version 4.6 and later.

> **Note**
>
> **The `user-managed-service-account()` method is only available for VMs running within GCP.**

Authenticate to a user-managed service account of a GCP virtual machine using the VM Metadata Server Method. AxoSyslog interacts with the internal GCP metadata server, which provides an OAuth2 token for authentication. You can attach the default service accounts as well.

metadata-url()

Type:	string
Default:	`http://metadata.google.internal/computeMetadata/v1/instance/service-accounts`

The URL of the metadata server. When specifying the port, use the `URL:port` format.

name()

Type:	string
Default:	`default`

Name of the service account to use.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	0

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

channel-args()

Type:	arrow list
Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

compression()

Type:	boolean
Default:	`no`

Available in AxoSyslog version 4.5.0 and later.

Description: Enables compression in gRPC requests. Although gRPC supports various compression methods, currently only deflate is supported (which is basically the same as gzip).

data()

Type:	string/template
Default:	`"${MESSAGE}"`

Description: The template to use as the data part of the Google Pub/Sub message.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

Note

Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ### Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

headers()

Type:	arrow list
Default:	empty

Available in AxoSyslog 4.8 and later.

Description: Adds custom gRPC headers to each RPC call. Version 4.8 supported only static header names and values. For example:

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "axo-stream"
  )
```

Starting with version 4.9, you can use templates and macros in the header values.

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "${HOST}"
  )
```

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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-alive()

Configures how AxoSyslog sends gRPC keepalive pings .

max-pings-without-data()

Type:	integer
Default:	

Description: The maximum number of gRPC pings that can be sent when there is no data/header frame to be sent. AxoSyslog won't send any pings after this limit. Set it to 0 disable this restriction and keep sending pings.

time()

Type:	number [milliseconds]
Default:	

Description: The period (in milliseconds) after which AxoSyslog sends a gRPC keepalive ping.

timeout()

Type:	number [milliseconds]
Default:	10

Description: The time (in milliseconds) AxoSyslog waits for an acknowledgement.

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property` : Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string` : Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message` : Drop the entire message silently, without logging the error.
- `silently-drop-property` : Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string` : Convert the property to string silently, without logging the error.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

project()

Type:	string/template
Default:	-

Description: The ID of the Google Cloud project where AxoSyslog sends the data. The Pub/Sub API must be enabled for the project.

proto-var()

Type:	template
Default:	-

Description: An alternative to using `attributes()` and `data()`. The advantage of `protovar()` is that the number of attributes it sends to the destination can vary between the messages, it doesn't have to be constant. That's useful is you're dynamically building your messages using FilterX.

response-action()

Type:	arrow list
Default:	Depends on the driver

Available in AxoSyslog version 4.11.0 and later.

Description: Fine-tunes how AxoSyslog behaves in case of different gRPC results. You can assign specific actions to the different gRPC results, for example:

```
response-action(
  not-found => disconnect
  unavailable => drop
)
```

The following gRPC results are supported:

- `aborted`
- `already-exists`
- `cancelled`
- `data-loss`
- `deadline-exceeded`
- `failed-precondition`
- `internal`
- `invalid-argument`
- `not-found`
- `ok`
- `out-of-range`
- `permission-denied`
- `resource-exhausted`
- `unauthenticated`
- `unavailable`
- `unknown`
- `unimplemented`

The following actions are available:

- `disconnect`
- `drop`
- `retry`
- `success`

retries()

Type: number (of attempts)

Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()` , AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

send-time-zone()

Accepted values:	name of the timezone, or the timezone offset
Default:	local timezone

Description: Specifies the time zone associated with the messages sent by `syslog-ng` , if not specified otherwise in the message or in the destination driver. For details, see Timezones and daylight saving.

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

service_endpoint()

Type:	string
Default:	`pubsub.googleapis.com:443`

The service endpoint for the Pub/Sub API where AxoSyslog sends the data. For a list of locational endpoints, see the official Pub/Sub APIs documentation .

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

time-zone()

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

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

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

topic()

Type:	string/template
Default:	-

Description: The name of the Google Pub/Sub topic where AxoSyslog sends the data.

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.**

url()

Type:	string
Default:	-

Description: An alias for `service_endpoint` .

worker-partition-autoscaling()

Type:	`yes` , `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)` , set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()` . Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()` .

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()` , `headers()` , or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that

contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()`.

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.13 Google Pub/Sub HTTP REST API

AxoSyslog can send data to Google Cloud Pub/Sub using its HTTP REST API .

Prerequisites

- AxoSyslog version 4.5.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `google-pubsub()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/google/google-pubsub.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- This feature requires a separate module. Install the `axosyslog-mod-cloud-auth` package on Debian/Ubuntu , or the `axosyslog-cloud-auth` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- A Google Pub/Sub subscription .
- An IAM service account that AxoSyslog uses for authentication.
- A Google Cloud project that has the Pub/Sub API enabled.

For details, see the Google Pub/Sub tutorial .

Configuration

To configure AxoSyslog, you'll need the name of the project and the topic where you want to send your data.

Minimal configuration:

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

destination d_pubsub {
  google-pubsub(
    project("syslog-ng-project")
    topic("syslog-ng-topic")
    auth(
      service-account(
        key("/path/to/service-account-key.json")
      )
    )
  );
};
```

Options

The following options are specific to the `google-pubsub()` destination. But since this destination is based on the `http()` destination, you can use the options of the `http()` destination as well if needed.

> Note: The `google-pubsub()` destination automatically configures some of these `http()` destination options as required by the Google Pub/Sub API.

attributes()

Type:	string
Default:	`"--scope rfc5424,all-nv-pairs --exclude MESSAGE"`

Description: A JSON object representing key-value pairs for the Pub/Sub Event, formatted as AxoSyslog value-pairs . By default, the `google-pubsub()` destination sends the RFC5424 fields as attributes. If you want to send different fields, override the default template. By default, the message part is sent in the `data()` option.

auth()

Options for cloud-related authentication. Currently only the GCP Service Account authentication is supported.

Specify the JSON file storing the key to the service account like this:

```
auth(
    service-account(
      key("/path/to/service-account-key.json")
    )
  )
```

service-account()

Authenticate to a service account using Service Account Key-Based Authentication. This method works both inside and outside GCP It uses a service account key generated and downloaded through the GCP IAM & Admin console. The long-term service account key is used to generate short-term tokens for authentication (also called self-signed JWT).

audience()

Type:	string
Default:	

> **Note**
>
> - When using the `google-pubsub()` destination, the `audience()` option is set to `https://pubsub.googleapis.com/google.pubsub.v1.Publisher`. Don't change it.
> - When using the `google-pubsub()-grpc` destination, you don't need to set the `audience()` option because the underlying gRPC API recognizes it automatically.
> - You cannot use `audience()` and `scope()` together.

key()

Type:	string (path)
Default:	

Path to the service account key.

scope()

Type:	string
Default:	

Available in AxoSyslog version 4.15 and later.

URL for a scope claim for authentication instead of audience . For example:

```
...
cloud-auth(
  gcp(
    service-account(
      key("/path/to/secret.json")
      scope("https://www.googleapis.com/auth/example-scope")
    )
  )
)
```

> **Note**
>
> **You cannot use `audience()` and `scope()` together.**

token-validity-duration()

Type:	integer (seconds)
Default:	`3600`

user-managed-service-account()

Available in AxoSyslog version 4.6 and later.

> **Note**
>
> **The `user-managed-service-account()` method is only available for VMs running within GCP.**

Authenticate to a user-managed service account of a GCP virtual machine using the VM Metadata Server Method. AxoSyslog interacts with the internal GCP metadata server, which provides an OAuth2 token for authentication. You can attach the default service accounts as well.

metadata-url()

Type:	string
Default:	`http://metadata.google.internal/computeMetadata/v1/instance/service-accounts`

The URL of the metadata server. When specifying the port, use the `URL:port` format.

name()

Type:	string
Default:	`default`

Name of the service account to use.

data()

Type:	string/template
Default:	`"${MESSAGE}"`

Description: The template to use as the data part of the Google Pub/Sub message.

project()

Type:	string
Default:	-

Description: The ID of the Google Cloud project where AxoSyslog sends the data. The Pub/Sub API must be enabled for the project.

topic()

Type:	string
Default:	-

Description: The name of the Google Pub/Sub topic where AxoSyslog sends the data.

service_endpoint()

Type:	string
Default:	`https://pubsub.googleapis.com`

The service endpoint for the Pub/Sub API where AxoSyslog sends the data. For a list of locational endpoints, see the official Pub/Sub APIs documentation .

8.14 graphite: Send 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 AxoSyslog 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 that 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 JSON parser) to AxoSyslog, like running a simple shell script collecting metrics and running it from cron regularly.

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

- On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-graphite` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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 .**

8.14.1 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.**

8.15 graylog2: Send 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. Version 3.21 and later also supports TLS-encrypted connection to the Graylog server.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

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 AxoSyslog side, configure the name or IP address of the host running Graylog.

```
destination d_graylog {
  graylog2(
    host("172.16.146.142")
    transport(tcp)
  );
};
```

If you parsed your messages using `syslog-n` g, 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, AxoSyslog 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 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 AxoSyslog 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")
  );
};
```

8.15.1 graylog2() destination options

The `graylog2()` destination has the following options:

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

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, AxoSyslog automatically sends multicast packets if a multicast destination address is specified. The `tcp` transport does not support multicasting.

8.16 hdfs: Store messages on the Hadoop Distributed File System (HDFS)

AxoSyslog 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).

Note the following limitations when using the AxoSyslog `hdfs` destination:

- Since AxoSyslog 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 AxoSyslog application to influence when the messages are actually written to disk. This means that AxoSyslog cannot guarantee that a message sent to HDFS is actually written to disk. When using flow-control, AxoSyslog 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.

Prerequisites

- AxoSyslog version 3.7 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `hdfs()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/hdfs/plugin.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-hdfs` package on Debian/Ubuntu , or the `axosyslog-java` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER`

error .

Declaration:

```
   @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.

```
   @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 AxoSyslog 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 the hdfs() destination.

- For the list of options, see HDFS destination options.

> **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.**

8.16.1 Prerequisites

To send messages from AxoSyslog to HDFS, complete the following steps.

Steps:

1. If you want to use the Java-based modules of AxoSyslog (for example, the Elasticsearch, HDFS, or Kafka destinations), you must compile AxoSyslog 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 AxoSyslog 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 AxoSyslog can access them. You must specify this directory in the AxoSyslog 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.

8.16.2 How AxoSyslog interacts with HDFS

The AxoSyslog application sends the log messages to the official HDFS client library, which forwards the data to the HDFS nodes. The way AxoSyslog interacts with HDFS is described in the following steps.

1. After AxoSyslog 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, AxoSyslog will repeatedly attempt to connect again after the period set in `time-reopen()` expires.

2. AxoSyslog checks if the path to the logfile exists. If a directory does not exist AxoSyslog automatically creates it. AxoSyslog creates the destination file (using the filename set in the AxoSyslog 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, AxoSyslog will write all incoming messages into the `hdfs` destination.

   > **Note**
   >
   > **When the `hdfs-append-enabled()` option is set to `true`, AxoSyslog 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 AxoSyslog application to influence when the messages are actually written to disk. This means that AxoSyslog cannot guarantee that a message sent to HDFS is actually written to disk. When using flow-control, AxoSyslog 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, AxoSyslog 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, AxoSyslog drops the message.

4. The AxoSyslog application closes the destination file in the following cases:

   - AxoSyslog is reloaded

- AxoSyslog is restarted
- The HDFS client returns an error.

5. If the file is closed and you have set an archive directory, AxoSyslog moves the file to this directory. If AxoSyslog cannot move the file for some reason (for example, AxoSyslog cannot connect to the HDFS NameNode), the file remains at its original location, AxoSyslog will not try to move it again.

8.16.3 Storing messages with MapR-FS

The AxoSyslog 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 AxoSyslog, 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.
    1. Download the libraries from the Maven Repository and Artifacts for MapR or get it from an already existing MapR installation.
    2. 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 AxoSyslog.

Example: Storing logfiles with MapR-FS

The following example defines an `hdfs` destination for MapR-FS using only the required parameters.

```
    @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")
        );
    };
```

8.16.4 Kerberos authentication with the hdfs() destination

Version 3.10 and later supports Kerberos authentication to authenticate the connection to your Hadoop cluster. AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog. For details, see the Kerberos documentation .
- You have installed and configured the Kerberos client packages on the host running AxoSyslog. (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"));
 };
```

8.16.5 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 line to the beginning of your AxoSyslog configuration:

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

client-lib-dir()

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

For the `hdfs` destination, include the path to the directory where you copied the required libraries (see Prerequisites), for example, `client-lib-dir("/opt/syslog-ng/lib/syslog-ng/java-modules/:/opt/hadoop/libs/")`.

disk-buffer()

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

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	string
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing,

and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()`.

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

hdfs-append-enabled()

Type:	`true` or `false`
Default:	false

Description: When `hdfs-append-enabled` is set to `true`, AxoSyslog will append new data to the end of an already existing HDFS file. Note that in this case, archiving is automatically disabled, and AxoSyslog will ignore the `hdfs-archive-dir` option.

When `hdfs-append-enabled` is set to `false`, the AxoSyslog 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, AxoSyslog 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).

> **Warning**
>
> **Before enabling the `hdfs-append-enabled` option, ensure that your HDFS server supports the `append` operation and that it is enabled. Otherwise AxoSyslog 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 AxoSyslog will move the closed log files. If AxoSyslog cannot move the file for some reason (for example, AxoSyslog cannot connect to the HDFS NameNode), the file remains at its original location. For example, `hdfs-archive-dir("/usr/hdfs/archive/")`.

> **Note**
>
> **When `hdfs-append-enabled` is set to `true`, archiving is automatically disabled, and AxoSyslog will ignore the `hdfs-archive-dir` option.**

hdfs-file()

Type:	string
Default:	N/A

Description: The path and name of the log file. For example, `hdfs-file("/usr/hdfs/mylogfile.txt")`. AxoSyslog checks if the path to the logfile exists. If a directory does not exist AxoSyslog automatically creates it.

`hdfs-file()` supports the usage of macros. This means that AxoSyslog 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, AxoSyslog 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 AxoSyslog 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`, AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**

Default: N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type: **string**

Default: N/A

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

`teardown()`

Type: **string**

Default: N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog 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.conf` 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 the 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 AxoSyslog 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 the 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 AxoSyslog 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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

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 AxoSyslog. 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-reap()

Accepted values:	number (seconds)
Default:	0 (disabled)

Description: The time to wait in seconds before an idle destination file is closed. Note that if `hdfs-archive-dir` option is set and `time-reap` expires, archiving is triggered for the affected file.

time-zone()

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

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

The timezone can be specified 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.**

8.17 java: Post messages over HTTP using Java

Version 3.7 of AxoSyslog 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 AxoSyslog. The current implementation has the following limitations:

- Only HTTP connections are supported, HTTPS is not.
- This destination requires Java. For an `http` destination that does not use Java, see http: Post messages over HTTP without Java.

Prerequisites

The Java HTTP module is no longer shipped with AxoSyslog: it isn't part of the `axosyslog-mod-java` (Debian/Ubuntu) or `axosyslog-java` (RHEL and compatible distributions) package. Use the http: Post messages over HTTP without Java destination instead, which is available in the `axosyslog-mod-http` / `axosyslog-http` package.

Declaration:

```
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.

```
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.

8.17.1 HTTP destination options

The `http` destination of AxoSyslog 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()`.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

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 AxoSyslog 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 AxoSyslog configuration (for example, because you have multiple Java-based destinations), AxoSyslog will merge every available paths to a single list.

For the `http` destination, include the path to the java modules of AxoSyslog, 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog 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.conf` 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 AxoSyslog version 3.7.2 and newer.

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

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 AxoSyslog. 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")
```

8.18 http: Post messages over HTTP without Java

Send log messages to web services over HTTP or HTTPS with the http() destination.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

AxoSyslog 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.

If the server returns a status code beginning with 2 (for example, 200), AxoSyslog assumes the message was successfully sent. For other response codes, see HTTP destination options. You can override the behavior of AxoSyslog using the `response-action()` option.

Prerequisites

- AxoSyslog version 3.8 or later.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Example: Client certificate authentication with HTTPS

```
destination d_https {
     http(
         [...]
         tls(
         ca-file("/<path-to-certificate-directory>/ca-crt.pem")
         ca-dir("/<path-to-certificate-directory>/")
         cert-file("/<path-to-certificate-directory>/server-crt.pem")
         key-file("/<path-to-certificate-directory>/server-key.pem")
             )
         [...]
     );
 };
```

Declaration:

```
destination d_http {
     http(
         url("<web-service-IP-or-hostname>")
         method("<HTTP-method>")
         user-agent("<USER-AGENT-message-value>")
         user("<username>")
         password("<password>")
     );
 };
```

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

Alternatively, you can leave the HTTP as-is, in which case the driver leaves the default `http_proxy` and `https_proxy` environment variables unmodified.

For more detailed information about these environment variables, see the libcurl documentation .

> **Note**
>
> **Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.**

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 AxoSyslog .

8.18.1 Batch mode and load balancing

Starting with version 3.18, you can send multiple log messages in a single HTTP request if the destination HTTP server supports that.

Batch size

The `batch-bytes()`, `batch-lines()`, `batch-timeout()`, and `batch-idle-timeout()` options of the destination determine how many log messages AxoSyslog sends in a batch. The `batch-lines()` option determines the maximum number of messages AxoSyslog puts in a batch in. This can be limited based on size and time:

- AxoSyslog sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. You can also set `batch-idle-timeout()` to close the batch if too much time elapses since the last message. That way the destination receives every message in a timely manner even if there are no more messages.
- AxoSyslog sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Formatting the batch

By default, AxoSyslog separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option.

If the target application or server requires a special beginning or ending to recognize batches, use the `body-prefix()` and `body-suffix()` options to add a beginning and ending to the batch. For example, you can use these options to create JSON-encoded arrays as POST payloads, which is required by a number of REST APIs. The body of a batch HTTP request looks like this:

```
value of body-prefix() option
log-line-1 (as formatted in the body() option)
log-line-2 (as formatted in the body() option)
....
log-line-n (the number of log lines is batch-lines(), or less if batch-timeout()
has elapsed or the batch would be longer than batch-bytes())
value of body-suffix() option
```

Example: HTTP batch mode

The following destination sends log messages to an Elasticsearch server using the bulk API . A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 10 seconds (10000 milliseconds).

```
destination d_http {
    http(url("http://your-elasticsearch-server/_bulk")
        method("POST")
        batch-lines(100)
        batch-bytes(512Kb)
        batch-timeout(10000)
        headers("Content-Type: application/x-ndjson")
        body-suffix("\n")
        body('{ "index":{} }
            $(format-json --scope rfc5424 --key ISODATE)')
    );
};
```

Load balancing between multiple servers

Starting with version 3.19, you can specify multiple URLs, for example, `url("site1" "site2")` . In this case, AxoSyslog sends log messages to the specified URLs in a load-balance fashion. This means that AxoSyslog sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order AxoSyslog has received

them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

> **Warning**
>
> **If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.**

Starting with version AxoSyslog version 3.22, you can use any of the following formats to specify multiple URLs:

```
url("server1", "server2", "server3"); # comma-separated strings
url("server1" "server2" "server3"); # space-separated strings
url("server1 server2 server3"); # space-separated within a single string
```

Example: HTTP load balancing

The following destination sends log messages to an Elasticsearch server using the bulk API , to 3 different ingest nodes. Each node is assigned a separate worker thread. A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 10 seconds (10000 milliseconds).

```
destination d_http {
    http(url("http://your-elasticsearch-server/_bulk" "http://your-second-ingest-
node/_bulk" "http://your-third-ingest-node/_bulk")
        method("POST")
        batch-lines(100)
        batch-bytes(512Kb)
        batch-timeout(10000)
        workers(3)
        headers("Content-Type: application/x-ndjson")
        body-suffix("\n")
        body('{ "index":{} }
            $(format-json --scope rfc5424 --key ISODATE)')
        persist-name("d_http-load-balance")
    );
};
```

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

Templates in the url()

Available in AxoSyslog version 4.5.0 and later.

In AxoSyslog, a template can only be resolved on a single message, because the same template might have different resolutions on different messages. As a batch consists of multiple messages, it's not trivial to decide which message should be used for the resolution.

When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()` option contains message-dependent templates, you must set the `worker-partition-key()` option to a template that contains all the message-dependent templates used in the `url()` option, otherwise the destination fails to start. AxoSyslog flushes the batch whenever the partition key changes, so every message of a batch generates the same URL.

For security reasons, all the templated contents in the `url()` option are URL-encoded automatically. The following parts of the URL cannot be templated:

- scheme
- host
- port
- user
- password

8.18.2 HTTP destination options

The `http` destination of AxoSyslog can directly post log messages to web services using the HTTP protocol. The `http` destination has the following options.

accept-encoding()

Type:	`"identity"` , `"gzip"` , `"deflate"` , `"all"`
Default:	N/A (disabled)

Description: Use `accept-encoding()` to request the server to compress the HTTP responses. (AxoSyslog doesn't currently use them, but they still contribute to network traffic.) To compress the messages sent by AxoSyslog, see the `content-compression()` option .

- If you want to accept multiple compression types, list them separated by commas inside the quotation mark.
- Use `"identity"` for no compression.
- To enable all available compression types (including no compression), use `"all"` .
- By default, AxoSyslog doesn't send the `Accept-Encoding:` header, and the received response isn't decompressed.

For example:

```
destination d_http_compressed{
  http(url("127.0.0.1:80") content-compression("deflate") accept-encoding("all"));
};
```

accept-redirects()

Type:	`yes` or `no`
Default:	

Description: Accept and follow redirect responses.

azure-auth-header()

See The Azure auth header plugin.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number

Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing

body()

Type:	string or template
Default:	

Description: The body of the HTTP request, for example, `body("${ISODATE} ${MESSAGE}")`. You can use strings, macros, and template functions in the body. If not set, it will contain the message received from the source by default.

body-prefix()

Accepted values:	string or template
Default:	none

Available in AxoSyslog version 3.18 and later.

Description: The string AxoSyslog puts at the beginning of the body of the HTTP request, before the log message. Starting with version 4.15, `body-prefix()` can include templates and macros as well. When using a template, note that:

- Literal dollar signs (`$`) used in `body-prefix()` must be escaped like `$$` .
- When using batching with a template, make sure to set the `worker-partition-key()` parameter appropriately to group similar messages together.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing

body-suffix()

Accepted values:	string
Default:	none

Description: The string AxoSyslog puts to the end of the body of the HTTP request, after the log message. Available in AxoSyslog version 3.18 and later.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing .

cloud-auth()

Authenticate to cloud-based services, for example, Azure and GCP, using service accounts.

azure()

Azure authentication currently supports authenticating to Azure Monitor, for example:

```
cloud-auth(
  azure(
    monitor(
      tenant-id("my-tenant-id")
      app-id("my-app-id")
      app-secret("my-app-secret")
    )
  )
)
```

To authenticate, you need to register a Microsoft Entra application . You'll need the Tenant ID, App ID, and App Secret of this application to configure the AxoSyslog destination.

app-id()

Type:	string
Default:	

Description: Application (client) ID of the Microsoft Entra application.

app-secret()

Type:	string
Default:	

Description: The Client secret of the Microsoft Entra application.

auth-url()

Type:	string
Default:	`https://login.microsoftonline.com`

Available in AxoSyslog 4.29 and later.

Description: The Microsoft Entra login host that AxoSyslog requests the OAuth2 access token from. AxoSyslog sends the token request to `<auth-url>/<tenant-id>/oauth2/v2.0/token`. Set this option to reach a national or sovereign cloud instead of the public Azure cloud. For example, for Azure Government, use `auth-url("https://login.microsoftonline.us")`.

scope()

Type:	string
Default:	`https://monitor.azure.com//.default`

Description: The scope to request for the access token.

tenant-id()

Type:	string
Default:	

Description: Directory (tenant) ID of the Microsoft Entra application.

gcp()

Authenticate to GCP service accounts. For example:

```
cloud-auth(
  gcp(
    user-managed-service-account(
      name("your-user@your-project.iam.gserviceaccount.com")
      metadata-url("your-metadata-server:8080")
    )
  )
)
```

service-account()

Authenticate to a service account using Service Account Key-Based Authentication. This method works both inside and outside GCP It uses a service account key generated and downloaded through the GCP IAM & Admin console. The long-term service account key is used to generate short-term tokens for authentication (also called self-signed JWT).

audience()

Type:	string
Default:	

> **Note**
>
> - When using the `google-pubsub()` destination, the `audience()` option is set to `https://pubsub.googleapis.com/google.pubsub.v1.Publisher`. Don't change it.
> - When using the `google-pubsub()-grpc` destination, you don't need to set the `audience()` option because the underlying gRPC API recognizes it automatically.
> - You cannot use `audience()` and `scope()` together.

key()

Type:	string (path)
Default:	

Path to the service account key.

scope()

Type:	string
Default:	

Available in AxoSyslog version 4.15 and later.

URL for a scope claim for authentication instead of audience . For example:

```
...
cloud-auth(
  gcp(
    service-account(
      key("/path/to/secret.json")
      scope("https://www.googleapis.com/auth/example-scope")
    )
  )
)
```

> **Note**
>
> **You cannot use `audience()` and `scope()` together.**

token-validity-duration()

Type:	integer (seconds)
Default:	`3600`

user-managed-service-account()

Available in AxoSyslog version 4.6 and later.

> **Note**
>
> **The `user-managed-service-account()` method is only available for VMs running within GCP.**

Authenticate to a user-managed service account of a GCP virtual machine using the VM Metadata Server Method. AxoSyslog interacts with the internal GCP metadata server, which provides an OAuth2 token for authentication. You can attach the default service accounts as well.

metadata-url()

Type:	string
Default:	`http://metadata.google.internal/computeMetadata/v1/instance/service-accounts`

The URL of the metadata server. When specifying the port, use the `URL:port` format.

name()

Type:	string
Default:	`default`

Name of the service account to use.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The AxoSyslog application uses this certificate to validate the certificate of the HTTPS server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cert-file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate (or a certificate chain) in PEM format, suitable as a TLS certificate, matching the private key set in the `key-file()` option. The AxoSyslog application uses this certificate to authenticate the AxoSyslog client on the destination server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that AxoSyslog 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 AxoSyslog. 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 AxoSyslog receives TLS-encrypted connections, the order of ciphers set on the AxoSyslog server takes precedence over the client settings.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

content-compression()

Type:	`"identity"` , `"gzip"` , `"deflate"`
Default:	`"identity"`

Description: Use `content-compression()` to compress the messages sent by AxoSyslog. Use `"identity"` for no compression. To accept compressed responses from the server, see the `accept-encoding()` option .

For example:

```
destination d_http_compressed{
  http(url("127.0.0.1:80") content-compression("deflate") accept-encoding("all"));
};
```

Note that `content-compression()` keeps the original, plain-text content if the compressed payload would be larger than the compressed payload. If your destination server accepts only compressed payload, enable the `force-content-compression(yes)` option as

well.

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, AxoSyslog separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option. Available in AxoSyslog version 3.18 and later.

For details on how this option influences HTTP batch mode, see Batch mode and load balancing

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

flush-lines()

Type:	number
Default:	Use global setting (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

flush-timeout() (DEPRECATED)

Type:	time in milliseconds
Default:	10000 [milliseconds]

This option is deprecated. Several destination drivers support the `batch-timeout()` option instead, which you can set at the destination level.

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends flushes to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `flush-timeout()` seconds.

force-content-compression()

Accepted values:	`yes` , `no`
Default:	`no`

Available in AxoSyslog 4.23 and later.

Description: If `content-compression()` is set to compress the messages, setting `force-content-compression(yes)` instructs AxoSyslog to always send the compressed payload, even when it's larger than the original, uncompressed content. Use `force-content-compression(yes)` if your destination server accepts only compressed payload.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

headers()

Type:	string list
Default:	see description

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>`

Starting with AxoSyslog 4.18, you can use templates in the headers. If you enable batching in the destination (`batch-lines()` or `batch-bytes()`) and use message-dependent templates in `headers()` , you must set the `worker-partition-key()` option, otherwise the destination fails to start. AxoSyslog flushes the batch whenever the partition key changes, so every message of a batch has the same header values. (In versions before 4.29, the header values were calculated from the first message of the batch.)

If you want to use literal dollar signs (`$`) in `headers()` , escape them like `$$` .

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog created rule is there, packets can flow, otherwise the port is closed.

```
source {
    network(transport(udp)
    hook-commands(
          startup("iptables -I LOGCHAIN 1 -p udp --dport 514 -j ACCEPT")
          shutdown("iptables -D LOGCHAIN 1")
        )
     );
};
```

key-file()

Accepted values:	Filename
Default:	none

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

The `http()` destination supports only unencrypted key files (that is, the private key cannot be password-protected).

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

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.

method()

Type:	`POST` or `PUT`
Default:	POST

Description: Specifies the HTTP method to use when sending the message to the server.

ocsp-stapling-verify

Accepted values:	`yes` , `no`
Default:	`no`

Available in AxoSyslog 4.0 and later.

Description: When OCSP stapling verification is enabled, AxoSyslog requests the server to send back its OCSP status. AxoSyslog verifies this status response using the trust store you have configured using the `ca-file()` , `ca-dir()` , or the `pkcs12-file()` options.

> **Note**
>
> **RFC 6961 multi-stapling and TLS 1.3-provided multiple responses are currently not validated, only the peer certificate is verified.**

Example configuration:

```
destination {
    http(url("https://example.com") method("POST") tls(peer-verify(yes) ocsp-
stapling-verify(yes)));
};
```

on-error()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property`: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string`: Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message`: Drop the entire message silently, without logging the error.
- `silently-drop-property`: Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string`: Convert the property to string silently, without logging the error.

password()

Type:	string
Default:	

Description: The password that AxoSyslog uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	`yes` or `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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()` , `ca-file()` , `cert-file()` , `cipher-suite()` , `key-file()` , `peer-verify()` , and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()` . Avoid mixing the two methods. 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:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

Alternatively, you can leave the HTTP as-is, in which case the driver leaves the default `http_proxy` and `https_proxy` environment variables unmodified.

> **Note**
>
> **Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.**

Example: the proxy() option in configuration

The following example illustrates including the `proxy()` option in your configuration.

```
destination {
   http( url("SYSLOG_SERVER_IP:PORT") proxy("PROXY_IP:PORT") method("POST"));
 };
```

python-http-header()

See The Python HTTP header plugin

response-action()

Type:	list
Default:	N/A (see below)

Description: Specifies what AxoSyslog does with the log message, based on the response code received from the HTTP server. If the server returns a status code beginning with 2 (for example, 200), AxoSyslog assumes the message was successfully sent. Otherwise, the action listed in the following table is applied. For status codes not listed in the following table, if the status code begins with 2 (for example, 299), AxoSyslog assumes the message was successfully sent. For other status codes, AxoSyslog disconnects. The following actions are possible:

- `disconnect` : Keep trying to resend the message indefinitely.
- `drop` : Drop the message without trying to resend it.
- `retry` : Retry sending the message for a maximum of `retries()` times (3 by default).
- `success` : Assume the message was successfully sent.

code	explanation	action
100	“Continue”	disconnect
101	“Switching Protocols”	disconnect
102	“Processing”	retry
103	“Early Hints”	retry
200	“OK”	success
201	“Created”	success
202	“Accepted”	success
203	“Non-Authoritative Information”	success
204	“No Content”	success
205	“Reset Content”	success

206	“Partial Content”	success
300	“Multiple Choices”	disconnect
301	“Moved Permanently”	disconnect
302	“Found”	disconnect
303	“See Other”	disconnect
304	“Not Modified”	retry
307	“Temporary Redirect”	disconnect
308	“Permanent Redirect”	disconnect
400	“Bad Request”	disconnect
401	“Unauthorized”	disconnect
402	“Payment Required”	disconnect
403	“Forbidden”	disconnect
404	“Not Found”	disconnect
405	“Method Not Allowed”	disconnect
406	“Not Acceptable”	disconnect
407	“Proxy Authentication Required”	disconnect
408	“Request Timeout”	disconnect
409	“Conflict”	disconnect
410	“Gone”	drop
411	“Length Required”	disconnect
412	“Precondition Failed”	disconnect
413	“Payload Too Large”	disconnect
414	“URI Too Long”	disconnect
415	“Unsupported Media Type”	disconnect
416	“Range Not Satisfiable”	drop
417	“Expectation Failed”	disconnect
418	“I’m a teapot”	disconnect

421	“Misdirected Request”	disconnect
422	“Unprocessable Entity”	drop
423	“Locked”	disconnect
424	“Failed Dependency”	drop
425	“Too Early”	drop
426	“Upgrade Required”	disconnect
428	“Precondition Required”	retry
429	“Too Many Requests”	disconnect
431	“Request Header Fields Too Large”	disconnect
451	“Unavailable For Legal Reasons”	drop
500	“Internal Server Error”	disconnect
501	“Not Implemented”	disconnect
502	“Bad Gateway”	disconnect
503	“Service Unavailable”	disconnect
504	“Gateway Timeout”	retry
505	“HTTP Version Not Supported”	disconnect
506	“Variant Also Negotiates”	disconnect
507	“Insufficient Storage”	disconnect
508	“Loop Detected”	drop
510	“Not Extended”	disconnect
511	“Network Authentication Required”	disconnect

To customize the action to take for a particular response code, use the arrow operator in the following format: `response-action(<response-code> => <action>)`. To customize multiple response code-action pairs, separate them with a comma, for example:

```
http(
    url("http://localhost:8080")
    response-action(418 => drop, 404 => retry)
);
```

response-adapter()

Type:	`openobserve` or `splunk`
Default:	N/A (disabled)

Available in AxoSyslog 4.27 and later.

Description: Some servers put the error data in the HTTP response body and not in the status code. The `response-adapter()` option lets AxoSyslog find these errors.

If `response-adapter()` finds an error, AxoSyslog sends the batch again. The `retries()` option sets the maximum number of tries. After the last try, AxoSyslog deletes the batch.

- `openobserve` : OpenObserve sends `200 OK` also for a request that has some errors. If you set `response-adapter(openobserve)` , AxoSyslog changes this response into an error. Then AxoSyslog can send the request again.
- `splunk` : This value processes the responses of Splunk HTTP Event Collector backends in the same way.

The `openobserve()` and `splunk-hec-event()` destinations set this option automatically.

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()` , AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

To handle HTTP error responses, if the HTTP server returns 5xx codes, AxoSyslog will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, AxoSyslog will drop the messages.

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.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in Macros of AxoSyslog. 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-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.

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.

url()

Type:	URL or list of URLs
Default:	http://localhost/

Description: Specifies the hostname or IP address and optionally the port number of the web service that can receive log data via HTTP. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://127.0.0.1:8000`

In case the server on the specified URL returns a redirect request, AxoSyslog automatically follows maximum 3 redirects. Only HTTP and HTTPS based redirections are supported.

Load balancing

Starting with version 3.19, you can specify multiple URLs, for example, `url("site1" "site2")` . In this case, AxoSyslog sends log messages to the specified URLs in a load-balance fashion. This means that AxoSyslog sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order AxoSyslog has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

> ### Warning
>
> **If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.**

Starting with version AxoSyslog version 3.22, you can use any of the following formats to specify multiple URLs:

```
url("server1", "server2", "server3"); # comma-separated strings
url("server1" "server2" "server3"); # space-separated strings
url("server1 server2 server3"); # space-separated within a single string
```

Templates in the url()

Available in AxoSyslog version 4.5.0 and later.

In AxoSyslog, a template can only be resolved on a single message, because the same template might have different resolutions on different messages. As a batch consists of multiple messages, it's not trivial to decide which message should be used for the resolution.

When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()` option contains message-dependent templates, you must set the `worker-partition-key()` option to a template that contains all the message-dependent templates used in the `url()` option, otherwise the destination fails to start. AxoSyslog flushes the batch whenever the partition key changes, so every message of a batch generates the same URL.

For security reasons, all the templated contents in the `url()` option are URL-encoded automatically. The following parts of the URL cannot be templated:

- scheme
- host
- port
- user
- password

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 AxoSyslog uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	`yes` or `no`
Default:	`no`

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation .

worker-partition-autoscaling()

Type:	`yes` , `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)` , set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()` . Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()` .

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()` , `headers()` , or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()` .

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.18.3 The Azure auth header plugin

This section describes the AxoSyslog application's Azure auth header plugin.

For more information about modules in AxoSyslog, see Using modules.

The Azure auth header plugin

The Azure auth header plugin is a signal-slot mechanism-based AxoSyslog module that generates authorization headers for applications that connect to Microsoft Azure.

Defining the Azure auth header plugin

You can define the Azure auth header plugin by the following:

```
azure-auth-header(
   method("POST")
   path("/api/logs")
   content-type("application/json")
   workspace-id("<workspace-id>")
   secret("<auth-secret>")
 )
```

Options

> **Note**
>
> **All these options are mandatory. They are used as input for the method that calculates the authorization header.**

8.18.4 The Python HTTP header plugin

This section describes the AxoSyslog application's Python HTTP header plugin.

For more information about modules in AxoSyslog, see Using modules.

The Python HTTP header plugin

The AxoSyslog application supports adding custom headers to HTTP requests using the Python programming language.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

> **Note**
>
> **Before you use the `python-http-header` plugin, make sure that your AxoSyslog appliance was compiled with Python support. If you installed AxoSyslog from a package, make sure that the subpackage containing Python support is also installed.**

Configuration

```
destination d_http {
   http(
     python_http_header(
       class("<class-name>")
       options("options-key1", "option-value1")
       options("options-key2", "option-value2")
       mark-errors-as-critical(no))
     url("http://127.0.0.1:8888")
   );
 };
```

Options used in the configuration:

- `class` : Mandatory option. It refers to the user's Python class that implements the `python-http-header` interface. It can be `mymodule.MyClass` if the `class MyClass` is put into a `mymodule.py` module, or simply `MyClass` if the user's code is provided inline in the configuration, using the `python { ... };` keyword.

> **Note**
>
> **If you put the class implementation into its own module, it should be put into a standard location, or made available with the `PYTHONPATH` environment variable.**

- `options("key" "value")` : Optional option. Multiple options can be specified at the same time. The AxoSyslog application will build a Python dictionary, which will be available in the `__init__` method.
- `mark-errors-as-critical(yes|no)` : Optional option. Its default value is `yes` . In case there is a Python error, this parameter decides if the HTTP destination will still try to send the request with the failed headers, or disconnect instead.

Defining the python-http-header() interface

You can define the Python interface with the following:

```
class TestCounter():
   def __init__(self, options):
     self.key = options["value"]

   def get_headers(self, body, headers):
     return ["header1: value1", "header2: value2"]

   def on_http_response_received(self, http_code):
     print("HTTP response code received: {}".format(http_code))
```

By default, when the `signal_http_header_request` is emitted by the HTTP module, the connected slot automatically executes the Python code.

> **Note**
>
> **If the plugin fails, the HTTP module does not send the HTTP request without the header items by default. If you want the HTTP module to try sending the request without the header items, disable the `mark-errors-as-critical` function.**

Methods used in the configuration:

- `__init__(self, options)` : Optional method. The options specified in the AxoSyslog configuration can be stored in the instance using this method.
- `get_headers(self, body, headers)` : Mandatory method. Returns a list of strings of form [`"header: value"` , ...]. The returned headers will be set for the outgoing HTTP request. The body contains the body of the HTTP request. The headers contain the current headers that the HTTP destination has already added to the request.
- `on_http_response_received(self, http_code)` : Optional method. If specified, AxoSyslog inserts the `http_code` of the previous response. This can be used to handle error (for example, for recreating auth headers, or dropping cache).

Example configuration for using the Python HTTP header plugin

The following example can be copy-pasted and used as a template for using the Python HTTP header plugin in your configuration.

```
   python {
    from syslogng import Logger

    logger = Logger()

    class TestCounter():
        def __init__(self, options):
            self.header = options["header"]
            self.counter = int(options["counter"])
            logger.debug(f"TestCounter class instantiated; options={options}")

        def get_headers(self, body, headers):
            logger.debug(f"get_headers() called, received body={body}, headers=
{headers}")

            response = ["{}: {}".format(self.header, self.counter)]
            self.counter += 1
```

```
            return response

        def on_http_response_received(self, http_code):
            self.counter += http_code
            logger.debug("HTTP response code received: {}".format(http_code))

        def __del__(self):
            logger.debug("Deleting TestCounter class instance")
    };

    source s_network {
      network(port(5555));
    };

    destination d_http {
        http(
            python_http_header(
                class("TestCounter")
                options("header", "X-Test-Python-Counter")
                options("counter", 11)
                # this means that syslog-ng will keep trying to send the http
request even when this module fails
                mark-errors-as-critical(no)
            )
            url("http://127.0.0.1:8888")
        );
    };

    log {
        source(s_network);
        destination(d_http);
        flags(flow-control);
    };
```

Warning

Although it is possible to configure multiple HTTP workers for AxoSyslog, the AxoSyslog application can only embed a single Python interpreter at the same time. As a result, if you configure more than one HTTP workers on your AxoSyslog application, the Python code will run in concurrent mode. To protect the state of the object, you may need to use locks.

For more information about using locks, see Introduction to the Python HTTP header .

8.19 kafka: Publish messages to Apache Kafka

The `kafka()` destination publishes log messages to the Apache Kafka message bus, where subscribers can access them.

`kafka()` is a configuration snippet of the AxoSyslog Configuration Library (SCL) that calls the `kafka-c()` destination and passes on every option unchanged. The two names behave identically, use whichever you prefer. You can find the source of this configuration snippet on GitHub .

> **Note**
>
> Earlier versions of syslog-ng had a separate Java implementation of the `kafka()` destination that used the official Java Kafka producer. AxoSyslog has removed that implementation: `kafka()` now always uses the C implementation, which builds on the librdkafka client library, scales better, and uses much less memory.
>
> The Java implementation used different option names. If you're migrating such a configuration, see Shifting from Java implementation to C implementation.

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

- This feature requires a separate module. Install the `axosyslog-mod-rdkafka` package on Debian/Ubuntu , or the `axosyslog-kafka` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

```
@include "scl.conf"

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 options.

```
@include "scl.conf"

destination d_kafka {
  kafka(
    bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
    topic("${HOST}")
  );
};
```

Options

The `kafka()` destination accepts the options of the `kafka-c()` destination. For the list of options, see Options of the kafka() destination's C implementation .

8.19.1 Prerequisites

To publish messages from AxoSyslog to Apache Kafka, complete the following steps.

Steps:

1. If you want to use the Java-based modules of AxoSyslog (for example, the Elasticsearch, HDFS, or Kafka destinations), you must compile AxoSyslog 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 AxoSyslog 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 AxoSyslog can access them. You must specify this directory in the AxoSyslog 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:
    1. Delete one of the files (for example, `slf4j-log4j12-<version-number>.jar`).
    2. Download a version that matches the version number of the other file (for example, 1.7.6) from the official SLF4J distribution .
    3. Copy the downloaded file into the directory of your Kafka library files (for example, `/opt/kafka/lib/`).

8.19.2 How AxoSyslog interacts with Apache Kafka

When stopping the AxoSyslog application, AxoSyslog 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 AxoSyslog 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 AxoSyslog `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 .

8.19.3 Kafka destination options

The `kafka` destination of AxoSyslog 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 AxoSyslog configuration:

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

client-lib-dir()

Type:	string
Default:	The AxoSyslog 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 AxoSyslog configuration (for example, because you have multiple Java-based destinations), AxoSyslog 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 AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog 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.conf` configuration file.

on-error()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `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 AxoSyslog application reads this file and passes the properties to the Kafka Producer. If a property is defined both in the AxoSyslog configuration file (`syslog-ng.conf`) and in the properties file, then AxoSyslog uses the definition from the AxoSyslog configuration file.

The AxoSyslog `kafka` destination supports all properties of the official Kafka producer. For details, see the Apache Kafka documentation .

The `kafka-bootstrap-servers` option is translated to the `bootstrap.servers` property.

For example, the following properties file defines the acknowledgment method and compression:

```
acks=all
 compression.type=snappy
```

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

sync-send()

Type:	`true` or `false`
Default:	false

Description: When `sync-send` is set to `true`, AxoSyslog sends the message reliably: it sends a message to the Kafka server, then waits for a reply. In case of failure, AxoSyslog repeats sending the message, as set in the `retries()` parameter. If sending the message fails for `retries()` times, AxoSyslog drops the message.

This method ensures reliable message transfer, but is very slow.

When `sync-send()` is set to `false`, AxoSyslog sends messages asynchronously, and receives the response asynchronously. In case of a problem, AxoSyslog cannot resend the messages.

This method is fast, but the transfer is not reliable. Several thousands of messages can be lost before AxoSyslog 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() .

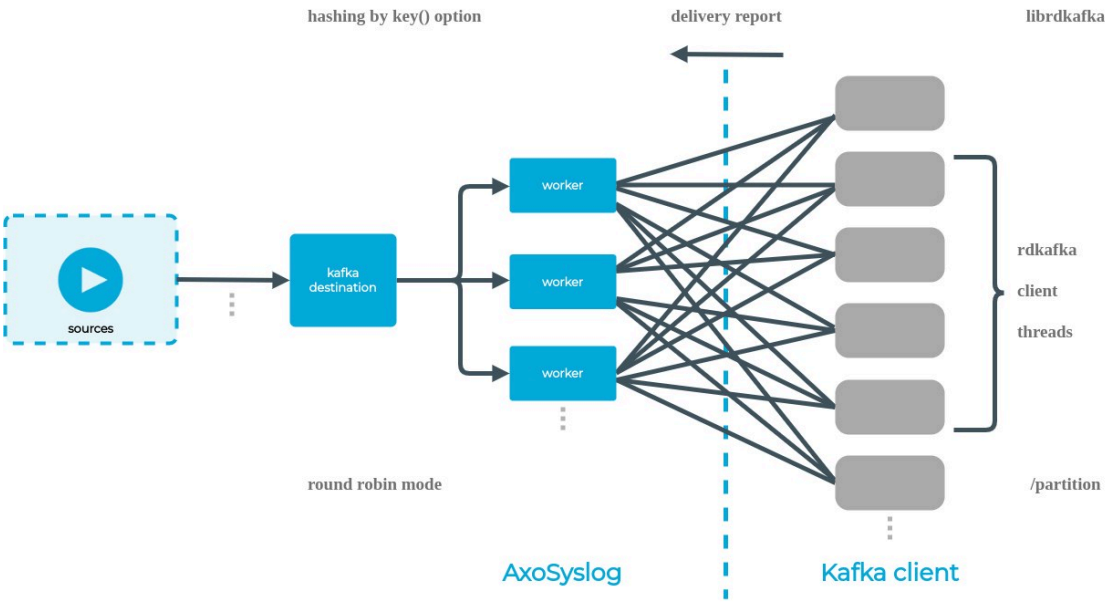
> **Note**
>
> **This option applies only to file and file-like destinations. Destinations that use specific protocols (for example, `network()` , or `syslog()`) ignore this option. For protocol-like destinations, use a template locally in the destination, or use the proto-template option.**

8.20 kafka-c(): Publish messages to Apache Kafka (C implementation)

AxoSyslog can directly publish log messages to the Apache Kafka message bus, where subscribers can access them.

The `kafka-c()` destination uses the librdkafka client library, which scales well, uses memory efficiently, and needs no Java runtime. It replaces the earlier Java implementation, which AxoSyslog has removed. If you're migrating a configuration written for the Java implementation, see Shifting from Java implementation to C implementation.

You can also configure this destination under the name `kafka()`, which is an SCL alias for `kafka-c()`. For details, see kafka: Publish messages to Apache Kafka .



Prerequisites

- AxoSyslog version 3.21 or later.
- This feature requires a separate module. Install the `axosyslog-mod-rdkafka` package on Debian/Ubuntu , or the `axosyslog-kafka` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

8.20.1 Shifting from Java implementation to C implementation

AxoSyslog has removed the Java implementation of the `kafka()` destination: both `kafka()` and `kafka-c()` now use the C implementation. If you're migrating a configuration written for the Java implementation, make the following changes.

- The `template()` option has been renamed `message()` .
- The `kafka-bootstrap-servers()` option has been renamed `bootstrap-servers()` .
- The `properties-file()` option has been removed. Set the Kafka producer properties in the `config()` option instead, which also replaces the old implementation's `option()` option and has a different syntax.
- The `client-lib-dir()` option has been removed. Remove it from the configuration file, the C implementation needs no Java libraries.
- The `@define kafka-implementation kafka-c` line is no longer needed. Remove it from the configuration file.
- The `topic()` option accepts templates. If you use a template, also set the `fallback-topic()` option, which AxoSyslog uses when the template resolves to an invalid topic name.

For more information, see Options of the kafka() destination's C implementation.

8.20.2 Before you begin

This section describes the prerequisites and restrictions for using the `kafka-c()` destination, and important information about the declaration of the destination.

Prerequisites and restrictions

- The `kafka-c()` destination uses the librdkafka client library .
- If you used the Java implementation before, see Shifting from Java implementation to C implementation.
- The AxoSyslog `kafka-c()` destination supports all properties of the official Kafka producer. For details, see the librdkafka documentation .
- For the list of options, see Options of the kafka() destination's C implementation.

Declaration:

```
kafka-c(
    bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
    topic("MYTOPIC")
);
```

Example: Sending log data to Apache Kafka

The following example defines a `kafka-c` destination, using only the required options.

```
destination d_kafka {
  kafka-c(
    bootstrap-servers("1.2.3.4:9092,192.168.0.2:9092")
    topic("MYTOPIC")
  );
};
```

You can also use the `kafka()` name, which is an SCL alias for `kafka-c()`. In that case, your configuration must contain `@include "scl.conf"`. For details, see kafka: Publish messages to Apache Kafka .

8.20.3 Flow control and the Kafka client

A AxoSyslog destination recognizes a message as sent when the message has been sent to the Kafka client, not when the Kafka server confirms its delivery.

If the Kafka client collects too many unsent messages, it will not accept any more messages from AxoSyslog. The AxoSyslog application detects this and stops sending messages to the Kafka client. Also, AxoSyslog's flow control starts functioning in the direction of the sources (for example, AxoSyslog will not read from the sources in that specific logpath).

You can specify a "high water mark" limit for the Kafka client with the `queue.buffering.max.messages` librdkafka property, which you can set in the `config()` option.

For more information about how the `kafka-c()` destination works with AxoSyslog, click here .

8.20.4 Options of the kafka() destination's C implementation

The C implementation of the `kafka()` destination of AxoSyslog can directly publish log messages to the Apache Kafka message bus, where subscribers can access them. The C implementation of the `kafka()` destination has the following options.

Required options:

The following options are required: `bootstrap-servers()`, `topic()`.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

> **Note**
>
> **The AxoSyslog configuration accepts `batch-lines()` with `sync-send()` set to both `"yes"` or `"no"`, but the option will only take effect if you set `sync-send()` to `"yes"`.**

> **Note**
>
> **If you set `sync-send()` to `"yes"`, the number you specify for `batch-lines()` affects how many messages AxoSyslog packs into once transaction.**

> **Note**
>
> When setting `batch-timeout()` , consider the value of the `transaction.timeout.ms` Kafka property. If in case of timeout (that is, if AxoSyslog does not receive `batch-lines()` amount of messages) the value of `batch-timeout()` exceeds the value of `transaction.timeout.ms` , AxoSyslog will not send out messages in time.
>
> For more information about the default values of the `transaction.timeout.ms` Kafka property, see the librdkafka documentation .

bootstrap-servers()

Type:	string
Default:	

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, `bootstrap-servers("127.0.0.1:2525,remote-server-hostname:6464")`

config()

Description: Sets the properties of the underlying librdkafka client. For example:

```
config(
   "acks" => "all"
   "compression.type" => "snappy"
)
```

The `bootstrap-servers()` option is translated to the `bootstrap.servers` property.

The AxoSyslog `kafka` destination supports all properties of the official Kafka producer. For details, see the librdkafka documentation .

The syntax of the config() option is the following:

```
config(
  “key1” => “value1”
  “key2” => “value2”
 )
```

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**

Default: no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type: string

Default: N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the

original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

fallback-topic()

Type:	string
Default:	N/A

Description: The topic to publish the message to if the name resolved from the `topic()` template isn't a valid Kafka topic name. Set this option whenever you use a template in `topic()` .

flush-timeout-on-reload()

Type:	integer in msec
Default:	1000

Description: When AxoSyslog reloads, the Kafka client will also reload. The `flush-timeout-on-reload()` option specifies the number of milliseconds AxoSyslog waits for the Kafka client to send the unsent messages. The unsent messages will be retained in `syslog-ng` 's own queue and AxoSyslog will continue sending them after reload. This works without disk-buffering, too.

flush-timeout-on-shutdown()

Type:	integer in msec
Default:	60000

Description: When AxoSyslog shuts down, the Kafka client will also shut down. The `flush-timeout-on-shutdown()` option specifies the number of milliseconds AxoSyslog waits for the Kafka client to send the unsent messages. Any messages not sent after the specified time will be lost. To avoid losing messages, we recommend you use the disk-buffer 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog created rule is there, packets can flow, otherwise the port is closed.

```
source {
    network(transport(udp)
    hook-commands(
          startup("iptables -I LOGCHAIN 1 -p udp --dport 514 -j ACCEPT")
          shutdown("iptables -D LOGCHAIN 1")
        )
     );
};
```

key()

Type:	template
Default:	empty string

Description: The key of the partition under which the message is published. You can use templates to change the topic dynamically based on the source or the content of the message, for example, `key("${PROGRAM}")`.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

local-time-zone()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Sets the timezone used when expanding filename and tablename templates.

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

message()

Type:	template or template function
Default:	`$ISODATE $HOST $MSGHDR$MSG`

Description: The message as published to Apache Kafka. You can use templates and template functions (for example, `format-json()`) to format the message, for example, `message("$(format-json --scope rfc5424 --exclude DATE --key ISODATE)")`.

For details on formatting messages in JSON format, see format-json.

> **Note**
>
> **In the Java implementation of the `kafka()` destination, this option was called `template()`. The C implementation doesn't accept `template()`.**

on-error()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property`: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.

- `fallback-to-string` : Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message` : Drop the entire message silently, without logging the error.
- `silently-drop-property` : Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string` : Convert the property to string silently, without logging the error.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog startup, set the `persist-name()` option of the duplicate drivers:

```
Error checking the uniqueness of the persist names, please override it with
persist-name option. Shutting down.
```

This error happens if you use identical drivers in multiple sources, for example, if you configure two file sources to read from the same file. In this case, set the `persist-name()` of the drivers to a custom string, for example, `persist-name("example-persist-name1")` .

poll-timeout()

Type:	integer in msec
Default:	1000

Description: Specifies the frequency your AxoSyslog queries the Kafka client about the amount of messages sent since the last `poll-timeout ()` . In case of multithreading, the first AxoSyslog worker is responsible for `poll-timeout()` .

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

send-time-zone()

Accepted values:	name of the timezone, or the timezone offset
Default:	local timezone

Description: Specifies the time zone associated with the messages sent by `syslog-ng`, if not specified otherwise in the message or in the destination driver. For details, see Timezones and daylight saving.

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

sync-send()

Type:	`true` or `false`
Default:	false

Description: When `sync-send` is set to `true`, AxoSyslog sends the message reliably: it sends a message to the Kafka server, then waits for a reply. In case of failure, AxoSyslog repeats sending the message, as set in the `retries()` parameter. If sending the message fails for `retries()` times, AxoSyslog drops the message.

This method ensures reliable message transfer, but is very slow.

When `sync-send()` is set to `false`, AxoSyslog sends messages asynchronously, and receives the response asynchronously. In case of a problem, AxoSyslog cannot resend the messages.

This method is fast, but the transfer is not reliable. Several thousands of messages can be lost before AxoSyslog recognizes the error.

> **Note**
>
> **If you want to use the `sync-send()` option set to `"yes"`, Axoflow recommends that you use librdkafka version 1.4.0 or higher, and a Kafka server with version number 0.11.0 or higher.**

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-zone()

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

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

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

topic()

Type:	template or template function
Default:	N/A

Description: The Kafka topic under which the message is published. You can use templates to set the topic dynamically based on the source or the content of the message, for example, `topic("${PROGRAM}")`. If the resolved topic name is invalid, AxoSyslog uses the topic set in the `fallback-topic()` option.

ts-format()

Type:	rfc3164, bsd, rfc3339, iso
Default:	rfc3164

Description: Override the global timestamp format (set in the global `ts-format()` parameter) for the specific destination. For details, see ts-format() .

> **Note**
>
> **This option applies only to file and file-like destinations. Destinations that use specific protocols (for example, `network()`, or `syslog()`) ignore this option. For protocol-like destinations, use a template locally in the destination, or use the proto-template option.**

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

Warning

`Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

Note

The workers are only responsible for formatting the messages that need to be delivered to the Kafka clients. Configure this option only if your Kafka clients have many threads and they do not receive enough messages.

Note

Kafka clients have their own threadpool, entirely independent from any AxoSyslog settings. The `workers()` option has no effect on this threadpool.

8.21 loggly: Send logs to 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.

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `loggly()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/loggly/loggly.conf on GitHub.

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})"));
     };
 }
```

8.21.1 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

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.

token()

Type:	string
Default:	

Description: Your Customer Token that you received from Loggly.

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, AxoSyslog automatically sends multicast packets if a multicast destination address is specified. The `tcp` transport does not support multicasting.

8.22 logmatic: Send logs to 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.

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `logmatic()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/logmatic/logmatic.conf on GitHub.

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})"));
     };
 }
```

8.22.1 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

8.23 Send messages to Falcon LogScale

AxoSyslog can send messages to Falcon LogScale using its Ingest Structured Data API . That way you don't have to parse the data on Falcon LogScale, because AxoSyslog already sends it in a structured format that LogScale understands and can show in a structured manner as separate columns. For a tutorial on using this destination in Kubernetes, see the From syslog-ng to LogScale: structured logs from any source blog post.

Prerequisites

- AxoSyslog version 4.3 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `logscale()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/logscale/logscale.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- Create an Ingest token for AxoSyslog to use in the `token()` option of the destination. This token is specific to a LogScale repository.

Ingest Structured Data API

The `logscale()` destination feeds LogScale via the Ingest Structured Data API .

Minimal configuration:

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

destination d_logscale {
  logscale(
    token("your-logscale-ingest-token")
  );
};
```

Options

The following options are specific to the `logscale()` destination. But since this destination is based on the `http()` destination, you can use the options of the `http()` destination as well if needed.

attributes()

Type:	string
Default:	`"--scope rfc5424 --exclude MESSAGE --exclude DATE --leave-initial-dot"`

Description: A JSON object representing key-value pairs for the LogScale Event, formatted as AxoSyslog value-pairs . By default, the `logscale()` destination sends the RFC5424 fields as attributes. If you want to send different fields, override the default template.

content-type()

Type:	string
Default:	`"application/json"`

Description: The content-type of the HTTP request.

extra-headers()

Type:	string
Default:	

Description: Extra headers for the HTTP request.

rawstring()

Type:	template
Default:	`${MESSAGE}`

Description: Accepts a template that you can use to format the LogScale event .

timestamp()

Type:	template
Default:	`${S_ISODATE}`

Description: The timestamp added to the LogScale event .

timezone()

Type:	string
Default:	

Description: The timezone of the event.

url()

Type:	string

Default: `"https://cloud.humio.com"`

Description: The URL of the LogScale Ingest API.

8.24 loki: Grafana Loki

The `loki()` destination sends your log data to Grafana Loki . Note that:

- AxoSyslog sends data using **gRPC**, HTTP transport is currently not supported. Since currently the Loki deployment in the free Grafana Cloud doesn't have gRPC enabled, you must use a self-deployed Grafana Loki instance.
- The message format is the same as documented for the Grafana Loki HTTP endpoint .

Sample configuration:

```
loki(
    url("localhost:9096")
    labels(
        "app" => "$PROGRAM",
        "host" => "$HOST",
    )

    workers(16)
    batch-timeout(10000)
    batch-lines(1000)
);
```

Prerequisites

- AxoSyslog version 4.4 or later.
- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Options

The `loki()` destination has the following options.

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

```
destination d_loki {
    loki(
      port(12345)
      auth(alts())
    );
  };
```

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_loki {
    loki(
      url("your-loki-server:12346")
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

batch-lines()

Type:	number
Default:	0

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

channel-args()

Type:	arrow list
Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

compression()

Type:	boolean
Default:	`no`

Available in AxoSyslog version 4.5.0 and later.

Description: Enables compression in gRPC requests. Although gRPC supports various compression methods, currently only deflate is supported (which is basically the same as gzip).

headers()

Type:	arrow list
Default:	empty

Available in AxoSyslog 4.8 and later.

Description: Adds custom gRPC headers to each RPC call. Version 4.8 supported only static header names and values. For example:

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "axo-stream"
  )
```

Starting with version 4.9, you can use templates and macros in the header values.

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "${HOST}"
  )
```

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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-alive()

Configures how AxoSyslog sends gRPC keepalive pings .

max-pings-without-data()

Type:	integer
Default:	

Description: The maximum number of gRPC pings that can be sent when there is no data/header frame to be sent. AxoSyslog won't send any pings after this limit. Set it to 0 disable this restriction and keep sending pings.

time()

Type:	number [milliseconds]
Default:	

Description: The period (in milliseconds) after which AxoSyslog sends a gRPC keepalive ping.

timeout()

Type:	number [milliseconds]
Default:	10

Description: The time (in milliseconds) AxoSyslog waits for an acknowledgement.

labels()

Type:	arrow list
Default:	

The labels applied to the message as they are sent to the destination. They must match the `[a-zA-Z_:][a-zA-Z0-9_:]*` regular expression, so labels can contain:

- ASCII letters,
- numbers, (but cannot begin with numbers)
- underscores (`_`), and
- colons (`:`).

Use the following format:

```
labels(
    "name_of_the_label_in_the_output" => "field-of-the-message",
)
```

For example:

```
    labels(
        "app" => "$PROGRAM",
        "host" => "$HOST",
    )
```

For details on using labels, see the Grafana Loki documentation .

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property`: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string`: Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message`: Drop the entire message silently, without logging the error.
- `silently-drop-property`: Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string`: Convert the property to string silently, without logging the error.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

response-action()

Type:	arrow list
Default:	Depends on the driver

Available in AxoSyslog version 4.11.0 and later.

Description: Fine-tunes how AxoSyslog behaves in case of different gRPC results. You can assign specific actions to the different gRPC results, for example:

```
response-action(
  not-found => disconnect
  unavailable => drop
)
```

The following gRPC results are supported:

- `aborted`
- `already-exists`
- `cancelled`

- `data-loss`
- `deadline-exceeded`
- `failed-precondition`
- `internal`
- `invalid-argument`
- `not-found`
- `ok`
- `out-of-range`
- `permission-denied`
- `resource-exhausted`
- `unauthenticated`
- `unavailable`
- `unknown`
- `unimplemented`

The following actions are available:

- `disconnect`
- `drop`
- `retry`
- `success`

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

send-time-zone()

Accepted values:	name of the timezone, or the timezone offset

Default:	local timezone

Description: Specifies the time zone associated with the messages sent by `syslog-ng`, if not specified otherwise in the message or in the destination driver. For details, see Timezones and daylight saving.

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

template()

Type:	template or template-function
Default:	`$ISODATE $HOST $MSGHDR$MSG`

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in Macros of AxoSyslog. For details on template functions, see Template functions of AxoSyslog.

tenant-id()

Type:	string
Default:	-

Description: Available in version 4.7 and newer. Sets the tenant ID for multi-tenant scenarios. For example:

```
loki(
    url("localhost:9096")
    labels(
        "app" => "$PROGRAM",
        "host" => "$HOST",
    )

    tenant-id("testTenant")
);
```

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

timestamp()

Type:	`current` , `received` , or `msg`
Default:	`current`

Description: Sets the timestamp to use for the messages sent to Loki. This is important because Loki accepts data only if their timestamp is monotonously increasing, out of order messages are rejected. The possible values for this option are:

- `current` : Use the timestamp when AxoSyslog processes the message in the output. This guarantees that the timestamp is monotonously increasing, but in some cases can significantly differ from the time when the message was generated.
- `msg` : Use the original timestamp of the message.
- `received` : Use the timestamp when AxoSyslog has received the message.

time-zone()

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

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

The timezone can be specified 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.

url()

Type:	string
Default:	`localhost:9095`

Description: The URL of the Loki endpoint, including the gRPC listen port of your Loki deployment.

worker-partition-autoscaling()

Type:	`yes` , `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)` , set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()` . Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()` .

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()` , `headers()` , or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()` .

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> ### Warning
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.25 mongodb(): Store 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.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-mongodb` package on Debian/Ubuntu , or the `axosyslog-mongodb` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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 AxoSyslog, use the `key()` parameter of the `value-pairs()` option.

The AxoSyslog `mongodb()` driver is compatible with MongoDB server version 1.4 and newer.

> **Note**
>
> Prior to version 4.0, AxoSyslog handled every data as strings, and allowed you to convert the strings into other types of data that only certain destinations data formats supported.
>
> Starting with AxoSyslog 4.0, each name-value pair is a (name, type, value) triplet, and several components of AxoSyslog have typing support. For details, see Components that support data types.

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")
         )
     );
 };
```

8.25.1 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:

bulk()

Type:	`yes` , `no`
Default:	`yes`

Available in AxoSyslog version 4.3.0 and newer.

Description: Enables bulk insert mode. If disabled, each messages is inserted individually.

> Note: Bulk sending is only efficient if you use a constant collection (without templates), or the used template does not lead to too many collections switching within a reasonable time range.

bulk-bypass-validation()

Type:	`yes` , `no`
Default:	`no`

Available in AxoSyslog version 4.3.0 and newer.

Description: If set to `yes` , it disables MongoDB bulk operations validation mode.

bulk-unordered()

Type:	`yes` , `no`
Default:	`no`

Available in AxoSyslog version 4.3.0 and newer.

Description: Enables unordered bulk operations mode.

collection()

Type:	template
Default:	messages

Description: The name of the MongoDB collection where the log messages are stored (collections are similar to SQL tables). You can use templates to change the collection dynamically based on the source or the content of the message, for example, collection("${HOST}").

> **Warning**
>
> **Hazard of data loss! The AxoSyslog application does not verify that the specified collection name does not contain invalid characters. If you specify a collection with an invalid name, the log messages sent to the MongoDB database will be irrevocably lost without any warning.**

disk-buffer()

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

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

Warning

Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

For MongoDB operations, AxoSyslog uses a one-minute timeout: if an operation times out, AxoSyslog assumes the operation has failed.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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 AxoSyslog 3.8 and later. Please refer to the MongoDB URI format documentation for detailed syntax.

value-pairs()

Type:	**parameter list of the `value-pairs()` option**
Default:	`scope("selected-macros" "nv-pairs")`

Description: The `value-pairs()` option creates structured name-value pairs from the data and metadata of the log message. For details on using `value-pairs()`, see Structuring macros, metadata, and other value-pairs.

> **Note**
>
> **Empty keys are not logged.**

Note

Prior to version 4.0, AxoSyslog handled every data as strings, and allowed you to convert the strings into other types of data that only certain destinations data formats supported.

Starting with AxoSyslog 4.0, each name-value pair is a (name, type, value) triplet, and several components of AxoSyslog have typing support. For details, see Components that support data types.

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

Warning

`Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

write-concern()

Type:	`unacked` , `acked` , or `majority`
Default:	`acked`

Available in AxoSyslog version 4.3.0 and newer.

Description: Sets the write concern mode of the MongoDB operations , for both bulk and single mode.

8.26 mqtt(): Send messages from a local network to an MQTT broker

AxoSyslog can use the `mqtt()` destination to publish messages to MQTT brokers.

The mqtt() destination builds on the MQTT protocol , and uses its “client ” and “broker ” entities.

> **Note**
>
> **The rest of this chapter and its sections build on your familiarity with the MQTT protocol, the concept of client and broker entities, and how these entities function within an MQTT system.**

Prerequisites

- AxoSyslog version 3.33 or later.
- This feature requires a separate module. Install the `axosyslog-mod-mqtt` package on Debian/Ubuntu , or the `axosyslog-mqtt` package on RHEL and compatible distributions . If the module isn’t installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

```
destination d_mqtt {
   mqtt(
     topic("<topic-name>"),
     address("tcp://localhost:<port-number>"),
     fallback_topic("<fallback-topic-name>")
   );
 }
```

Example: Using the mqtt() destination in your configuration

The following example illustrates a `mqtt()` destination configured to fetch messages from the `localhost:4444` address, and send them to the broker running on `localhost:4445`, using the `mqtt test/test` topic.

```
   @version: 3.32
    @include "scl.conf"

      source s_net {
        network(port(4444)
      );
    };

      destination d_mqtt {
        mqtt(topic("test/test"), address("tcp://localhost:4445"),
fallback_topic("test/test")
      );
    };

    log {
      source(s_net);
      destination( d_mqtt);
    };
```

8.26.1 Prerequisites to using the mqtt() destination

Using the current implementation of the `mqtt()` destination has the following prerequisites:

- Installing the `eclipse-paho-mqtt-c` library.

> **Note**
>
> **The default package manager for some Linux operating systems contains the `eclipse-paho-mqtt-c` library, but depending on your OS, you may have to install the library manually. For more information about how you can download and install the `eclipse-paho-mqtt-c` library, see Eclipse Paho on the Eclipse Foundation website.**

- Having a `broker` entity in a functional MQTT system.

> **Note**
>
> **In your configuration, you will specify the `broker` entity of your MQTT system in the `address()` option of your `mqtt()` source.**

8.26.2 Limitations to using the mqtt() destination

Using the `mqtt()` destination of AxoSyslog has the following limitations:

- You can only use the `mqtt()` destination with AxoSyslog version 3.33 or higher.
- You cannot use the `mqtt()` destination without installing the the `eclipse-paho-mqtt-c` library.

  For more information about how you can download and install the `eclipse-paho-mqtt-c` library, see Eclipse Paho on the Eclipse Foundation website.
- The current implementation of the `mqtt()` destination supports versions 3.1 and 3.1.1 of the MQTT protocol.

8.26.3 Options of the mqtt() destination

The `mqtt()` destination has the following options.

Required options: `address()`, `fallback-topic()`, and `topic()`.

address()

Type:	string
Default:	`tcp://localhost:1883`
Required:	yes

Description: Specifies the hostname or IP address, and the port number of the MQTT broker to which AxoSyslog will send the log messages.

Syntax: `<protocol type>://<host>:<port>`

Supported protocol types: `TCP`, `WS`, `SSL` and `WSS`.

client-id()

Type:	string
Default:	`syslog-ng-source-{topic option}`
Required:	no

Description: The `client-id()` is used to identify the client to the MQTT server, which stores session data for each client. The session data can contains information regarding which message has been sent, received. It is not possible to set the `client-id()` to an empty string. To always start a new session see the `cleansession()` option.

cleansession()

Type:	`yes` or `no`
Default:	`no`

Description: This option instruments the MQTT broker to clean the session data when connecting. The session data contains information about which message was processed.

fallback-topic()

Type:	string
Default:	N/A

Description: Required option when using templates in the `topic()` option.

If the resolved `topic()` template is not a valid topic, AxoSyslog will use the `fallback-topic()` option to send messages.

> **Note**
>
> **If instead of strings, you use actual templates (that is, a macro like `${MESSAGE}`, or a template function like `$(format-json)`) in the `topic()` option, configuring the `fallback-topic()` option is required.**

> **Note**
>
> **Occasionally, the reason why AxoSyslog cannot post messages to the configured `topic()` is that the topic contains invalid characters that originate from templates.**

http-proxy()

Type:	URL
Default:	N/A

Description: Specifies HTTP/HTTPS proxy for WebSocket connections.

keep-alive()

Type:	positive integer number (in seconds)

Default:	`60`

Description: Specifies the number of seconds that AxoSyslog keeps the connection between the broker and clients open in case there is no message traffic. When `keep-alive()` number of seconds pass, the connection is terminated, and you have to reconnect.

On the MQTT side, the keep alive function provides a workaround method to access connections that are still open.

password()

Type:	string
Default:	N/A

Description: The password used to authenticate on the MQTT broker.

qos()

Type:	number
Default:	`0`
Possible values:	`0` - at most once (the fastest option) `1` - at least once (a much slower option than `0`) `2` - exactly once (the slowest option)

Description: The Quality of Service (QoS) level in MQTT messaging is an agreement between sender and receiver on the guarantee of delivering a message.

template()

Type:	string
Default:	`$ISODATE $HOST $MSGHDR$MSG`

Description: Specifies the message template that AxoSyslog sends to the MQTT broker.

If you want to use macros in templates, see Macros of AxoSyslog.

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.

The following options are relevant for the `mqtt()` `tls()` block: `ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, `ssl-version()`, `use-system-cert-store()`.

topic()

Type:	string or template
Default:	N/A

Description: Required option. Specifies the MQTT topic.

> **Note**
>
> The current implementation of the `mqtt()` destination does not support using the following characters for topic names:
>
> - `$`
> - `+`
> - `#`

username()

Type:	string
Default:	N/A

Description: The username used to authenticate on the MQTT broker.

8.26.4 Possible error messages you may encounter while using the mqtt() destination

While using the `mqtt()` destination, you may encounter issues and corresponding error messages originating from the MQTT system. The following table contains the error messages you may encounter, the possible reasons behind them, and potential workaround methods.

Complete error message	Possible reason(s)	Possible solution(s)
`ERROR, while init threaded dest.`	The AxoSyslog application will not start.	You can try the following methods: • Restart AxoSyslog. • Stop some of the programs running on your computer. • Restart your computer, and then restart AxoSyslog.
`mqtt: the topic() argument is required for mqtt destinations.`	The `topic()` option is not set in your configuration. The AxoSyslog application will not start.	Set the missing `topic()` option in your configuration, then restart .
`The mqtt destination does not support the batching of messages, ...`	Your configuration may contain the `batch-timeout()` and / or `batch-lines()` options, which are not supported by the `mqtt()` destination. The AxoSyslog application will not start.	If your configuration contains the `batch-timeout()` and / or `batch-lines()` options, remove them from your configuration, and restart .
`Disconnected during publish!`	The AxoSyslog application can not send the message, because AxoSyslog disconnected from the broker. By default, AxoSyslog attempts to reconnect to the broker and send the messages 3 times.	If AxoSyslog fails all 3 attempts to reconnect to the broker and send the messages, you can try checking your configuration or restarting your MQTT system with AxoSyslog as a client.

`Max message inflight! (publish)`	The AxoSyslog application can not send the message due to the `max message inflight` broker response code (which signals that the broker has received too many messages, and it needs more time to process them). The AxoSyslog application will attempt to resend the message.	Wait until the broker can process the in-flight messages and AxoSyslog can attempt to resend the message.
`Failure during publishing!`	The AxoSyslog application can not send the message due to the `failure` broker response code. The AxoSyslog application will attempt to resend the message.	N/A
`Error during publish!`	The AxoSyslog application can not send the message, and drops it. Possible reason: `bad_utf8_string (topic), NULL parameter`. That is, the most probable reasons behind this issue are either that the topic name in your configuration is not correct, or that the message field is empty.	You can try the following methods: • Modify the name of the `topic()` option in your configuration. • Make sure that the message field is not empty.
`Disconnected while waiting the response!`	The AxoSyslog application has sent the message, but the client disconnected from the broker before AxoSyslog received the response. The AxoSyslog application will attempt to reconnect, or to resend the message.	The AxoSyslog application will attempt to reconnect to the broker and send the in-flight message. If the reconnect attempt fails, AxoSyslog will resend the message.
"Error while waiting the response!"	The AxoSyslog application can not get any response from the broker, due to the `failure` broker response code. The AxoSyslog will attempt to resend the message.	In this case, you will receive a further error message, depending on what the problem is. Wait for the second error message for more information about how you can proceed.

"Error constructing topic ..."	Due to an issue with the configured topic template, the `mqtt()` destination will use the `fallback-topic()` option instead.	N/A
`mqtt dest: topic name is illegal, it can't be empty`	This error message is related to the `"Error constructing topic ..."` error message. In this case, the topic template returns a `0` length string. As a result, the `mqtt()` destination will use the `fallback-topic()` option instead.	N/A
`Error connecting mqtt client ...`	The AxoSyslog application can not connect to broker, and it will attempt to reconnect later.	If the issue persists, you can try the following: • Update your `eclipse-paho-mqtt-c` library. • Restart AxoSyslog.
`Error creat mqtt client ...`	The AxoSyslog application encountered an error while creating the MQTT client, and it will attempt to create it later. Possible reasons: • There is a wrong `address()` set in your configuration. • The broker is not running.	You can try the following methods: • Check the `address()` option in your configuration, and modify if necessary. • Check if the specified broker is running by connecting to it manually, and then sending the broker a message.

8.27 network: Send 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 AxoSyslog 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, AxoSyslog sends messages using the TCP protocol to port 514.

Example: Using the network() driver

TCP destination that sends messages to `10.1.2.3`, port `1999`:

```
destination d_tcp { network("10.1.2.3" port(1999)); };
```

If name resolution is configured, you can use the hostname of the target server as well.

```
destination d_tcp { network("target_host" port(1999)); };
```

TCP destination that sends messages to the `::1` IPv6 address, port `2222`.

```
destination d_tcp6 {
     network(
         "::1"
         port(2222)
         transport(tcp)
         ip-protocol(6)
         );
 };
```

To send messages using the IETF-syslog message format without using the IETF-syslog protocol, enable the `syslog-protocol` flag. (For details on how to use the IETF-syslog protocol, see syslog() destination options.)

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

Batching writes with flush-lines()

Available in AxoSyslog 4.26 and later.

When a `network()` or `tcp()` destination uses a TCP or TLS transport, AxoSyslog writes several messages to the transport with a single operation instead of one per message. It accumulates up to `flush-lines()` formatted messages and sends them together: over TCP as one `writev()` system call, and over TLS as few `SSL_write()` calls as fit into a TLS record. Fewer system calls per message reduces overhead and can increase throughput. As with `flush-lines()` on any destination, larger batches trade a small amount of latency for that throughput, because a message can wait for the batch to fill (or for the queue to empty) before it is sent.

This applies only to the stream-based transports. UDP destinations are unaffected, because each datagram already carries exactly one message, and the `syslog()` destination is unaffected as well, because it frames every message individually.

Controlling the behavior with the configuration version

The global default of `flush-lines()` is `100`, so batching is enabled by default once your configuration declares `@version: 4.26` (or later). Configurations that declare an older version keep the previous one-write-per-message behavior, so that upgrading AxoSyslog does not silently change how an existing configuration sends messages.

If your configuration declares a version older than `4.26` and does not set `flush-lines()` on the destination, AxoSyslog logs a warning once and keeps the pre-4.26 behavior. To adopt batching and stop the warning, either raise the configuration version to `4.26` or set `flush-lines()` explicitly on the destination. Setting `flush-lines()` explicitly always takes effect, regardless of the configuration version.

To keep (or restore) one write per message, set `flush-lines(1)`:

```
destination d_tcp { network("10.1.2.3" port(1999) flush-lines(1)); };
```

8.27.1 network() destination options

The `network()` driver sends messages to a remote host (for example, a 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:

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

close-on-input()

Type:	`yes` or `no`
Default:	yes

Description: By default, AxoSyslog closes destination sockets if it receives any input from the socket (for example, a reply). If this option is set to no, AxoSyslog just ignores the input, but does not close the socket.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

failover()

Description: Available only in AxoSyslog 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, AxoSyslog waits for the a server before switching to the next failover server is set in the `time-reopen()` option.

If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

> **Warning**
>
> **The failover servers must be accessible on the same port as the primary server.**

failback()

Description: Available only in AxoSyslog version 3.17 and later.

When AxoSyslog 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, AxoSyslog behaves as follows:

- `round-robin mode` : If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

  In the following example AxoSyslog 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, AxoSyslog attempts to return to the primary server.

  After AxoSyslog 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, AxoSyslog tries to connect the primary server using the last probe.

  > **Note**
  >
  > **AxoSyslog 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.**

  In the following example AxoSyslog 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

Example: Configuring failover servers

In the following example AxoSyslog handles the logservers in round-robin fashion if the primary logserver becomes uneccassible (therefore `failback()` option is not set).

```
   destination demo_failover_roundrobin{
          syslog("primary-logserver.example.com"
                failover(
                      servers("first-failover-logserver.example.com", "second-
failover-logserver.example.com")
                )
          transport("tcp")
          port(514)
          time-reopen(60)
          );
    };
```

In the following example AxoSyslog 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 demo_failover_returntoprimary{
          syslog("primary-logserver.example.com"
                failover(
                      servers("first-failover-logserver.example.com", "second-
failover-logserver.example.com")
                      failback(
                            tcp-probe-interval(120)
                            successful-probes-required(3)
                      )
                )
          transport("tcp")
          port(514)
          time-reopen(60)
          );
    };
```

> **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 (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

UDP and other datagram transports write one message per packet.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 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 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 AxoSyslog 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 AxoSyslog.

> ### 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 AxoSyslog 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 AxoSyslog do not comply with the latest RFC which was published after the release of AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 using AxoSyslog 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.

The maximum size of spoofed datagrams in udp() destinations is set to 1024 bytes by default. To change the maximum size, use the `spoof-source-max-msglen()` option.

> **Note**
>
> Anything above the size of the maximum transmission unit (MTU), which is 1500 bytes by default, is not recommended because of fragmentation.
>
> The maximum datagram in IP protocols (both IPv4 and IPv6) is 65535 bytes including the IP and UDP headers. The minimum size of the IPv4 header is 20 bytes, the IPv6 is 40 bytes, and the UDP is 8 bytes.

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), AxoSyslog 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 AxoSyslog waits for identical messages.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

> **Warning**
>
> 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 AxoSyslog 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.

Warning

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 AxoSyslog 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.

Warning

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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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, AxoSyslog 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.**

8.28 Send messages to OpenObserve

Send log messages to OpenObserve with the openobserve-log() destination.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

AxoSyslog can send messages to OpenObserve using its Logs Ingestion - JSON API . This API accepts multiple records in batch in JSON format.

Prerequisites

- AxoSyslog version 4.5.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `openobserve-log()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/openobserve/openobserve.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- An OpenObserve account for AxoSyslog, or

- a self-hosted OpenObserve deployment .

Configuration

To configure AxoSyslog, you'll need the username, password, the name of your organization, and the name of the OpenObserve stream where you want to send your data.

Minimal configuration:

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

destination d_openobserve {
  openobserve-log(
    url("http://your-openobserve-endpoint")
    organization("your-organization")
    stream("your-example-stream")
    user("root@example.com")
    password("V2tsn88GhdNTKxaS")
  );
};
```

Example configuration:

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

destination d_openobserve {
  openobserve-log(
    url("https://api.openobserve.ai")
    port(443)
    organization("your-organization")
    stream("your-example-stream")
    user("root@example.com")
    password("V2tsn88GhdNTKxaS")
  );
};
```

Options

The following options are specific to the `openobserve-log()` destination. But since this destination is based on the `http()` destination, you can use the options of the `http()` destination as well if needed.

> Note: The `openobserve-log()` destination automatically configures some of these `http()` destination options as required by the OpenObserve Ingest API.

organization()

Type:	string
Default:	`"default"`

Description: The name of the OpenObserve organization where AxoSyslog sends the data.

password()

Type:	string
Default:	-

Description: The password for the username specified in the `user()` option.

port()

Type:	integer
Default:	`5080`

Description: The port number of the server.

record()

Type:	string
Default:	`"--scope rfc5424 --exclude DATE --key ISODATE @timestamp=${ISODATE}"`

Description: A JSON object representing key-value pairs sent to OpenObserve, formatted as AxoSyslog value-pairs . By default, the `openobserve-log()` destination sends the RFC5424 fields as attributes. If you want to send different fields, override the default content of the `record()` field.

response-adapter()

Type:	`openobserve` or `splunk`
Default:	N/A (disabled)

Available in AxoSyslog 4.27 and later.

Description: Some servers put the error data in the HTTP response body and not in the status code. The `response-adapter()` option lets AxoSyslog find these errors.

If `response-adapter()` finds an error, AxoSyslog sends the batch again. The `retries()` option sets the maximum number of tries. After the last try, AxoSyslog deletes the batch.

- `openobserve` : OpenObserve sends `200 OK` also for a request that has some errors. If you set `response-adapter(openobserve)` , AxoSyslog changes this response into an error. Then AxoSyslog can send the request again.
- `splunk` : This value processes the responses of Splunk HTTP Event Collector backends in the same way.

The `openobserve()` and `splunk-hec-event()` destinations set this option automatically.

stream()

Type:	string
Default:	`"default"`

Description: The OpenObserve stream where AxoSyslog sends the data, for example, `your-example-stream` .

user()

Type:	string
Default:	-

Description: The username of the account, for example, `root@example.com`.

url()

Type:	string
Default:	-

Description: The base URL of the OpenObserve Ingest API. The actual URL is constructed from the base URL and some other options of the destination:
`url():port()/api/organization()/stream()/_json`

8.29 opensearch: Send messages to OpenSearch

The `opensearch()` destination can directly post log messages to OpenSearch using its HTTP endpoint.

HTTPS connection, as well as password- and certificate-based authentication is supported. The content of the events is sent in JSON format.

Prerequisites

- AxoSyslog version 4.4 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `opensearch()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/opensearch/opensearch.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

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

d_opensearch {
    opensearch(
        index("<opensearch-index-to-store-messages>")
        url("https://your-opensearch-endpoint:9200/_bulk")
    );
};
```

Example: Sending log data to OpenSearch

The following example defines an `opensearch()` destination, with only the required options.

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

destination opensearch {
    opensearch(
        index("<name-of-the-index>")
        url("http://my-elastic-server:9200/_bulk")
    );
};

log {
    source(s_file);
    destination(d_opensearch_http);
    flags(flow-control);
};
```

The following example uses mutually-authenticated HTTPS connection, templated index, and also sets some other options.

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

destination opensearch_https {
    opensearch(
        url("https://node01.example.com:9200/_bulk")
        index("test-${YEAR}${MONTH}${DAY}")
        time-zone("UTC")
        workers(4)
        batch-lines(16)
        timeout(10)
        tls(
            ca-file("ca.pem")
            cert-file("syslog_ng.crt.pem")
            key-file("syslog_ng.key.pem")
            peer-verify(yes)
        )
    );
};
```

8.29.1 Batch mode and load balancing

The `opensearch()` destination automatically sends multiple log messages in a single HTTP request, increasing the rate of messages that your Elasticsearch deployment can consume. For details on adjusting and fine-tuning the batch mode, see the following section.

Batch size

The `batch-bytes()`, `batch-lines()`, `batch-timeout()`, and `batch-idle-timeout()` options of the destination determine how many log messages AxoSyslog sends in a batch. The `batch-lines()` option determines the maximum number of messages AxoSyslog puts in a batch in. This can be limited based on size and time:

- AxoSyslog sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()`. You can also set `batch-idle-timeout()` to close the batch if too much time elapses since the last message. That way the destination receives every message in a timely manner even if there are no more messages.
- AxoSyslog sends the batch if the total size of the messages in the batch reaches `batch-bytes()` bytes.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-bytes()`, `batch-lines()`, `batch-timeout()` options.

Example: HTTP batch mode

In the following example, a batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
destination d_opensearch {
     opensearch(
         url("http://your-server:9200/_bulk")
         index("<index-to-store-messages>")
         batch-lines(100)
         batch-bytes(512Kb)
         batch-timeout(10000)
     );
 };
```

Load balancing between multiple indexers

Starting with version 3.19, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, AxoSyslog sends log messages to the specified URLs in a load-balance fashion. This means that AxoSyslog sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order AxoSyslog has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

> **Warning**
>
> **If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.**

Starting with version AxoSyslog version 3.22, you can use any of the following formats to specify multiple URLs:

```
url("server1", "server2", "server3"); # comma-separated strings
url("server1" "server2" "server3"); # space-separated strings
url("server1 server2 server3"); # space-separated within a single string
```

Example: HTTP load balancing

The following destination sends log messages to 3 different indexer nodes. Each node is assigned a separate worker thread. A batch consists of 100 messages, or a maximum of 512 kilobytes, and is sent every 20 seconds (20000 milliseconds).

```
   destination d_opensearch {
        opensearch(
            url("http://your-elasticsearch-server1:9200/_bulk" "http://your-
elasticsearch-server2:9200/_bulk" "http://your-elasticsearch-server3:9200/_bulk")
            batch-lines(100)
            batch-bytes(512Kb)
            batch-timeout(20000)
            persist-name("opensearch-load-balance")
        );
    };
```

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.29.2 opensearch() destination options

The `opensearch` destination of AxoSyslog can directly post log messages to an OpenSearch deployment using the OpenSearch Bulk API over the HTTP and Secure HTTP (HTTPS) protocols. The `opensearch` destination has the following options. The required options are: `index()` and `url()`.

This destination is available in AxoSyslog version 4.4 and later.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

For details on how this option influences batch mode, see Batch mode and load balancing.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	25

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

For details on how this option influences batch mode, see Batch mode and load balancing.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

For details on how this option influences batch mode, see Batch mode and load balancing.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

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

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

Declaration:

```
destination d_opensearch {
     opensearch(
         url("http://your-server:9200/_bulk")
         index("example-index")
         tls(
             ca-dir("dir")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

ca-file()

Accepted values:	Filename
Default:	none

Description: Name of a file that contains an X.509 CA certificate (or a certificate chain) in PEM format. The AxoSyslog application uses this certificate to validate the certificate of the HTTPS server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

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

Declaration:

```
destination d_opensearch {
     opensearch(
         url("http://your-server:9200/_bulk")
         index("example-index")
         tls(
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cert-file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate (or a certificate chain) in PEM format, suitable as a TLS certificate, matching the private key set in the `key-file()` option. The AxoSyslog application uses this certificate to authenticate the AxoSyslog client on the destination server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

cipher-suite()

Accepted values:	Name of a cipher, or a colon-separated list
Default:	Depends on the OpenSSL version that AxoSyslog 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 AxoSyslog. 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 AxoSyslog receives TLS-encrypted connections, the order of ciphers set on the AxoSyslog server takes precedence over the client settings.

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

custom-id()

Accepted values:	string
Default:	empty string

Description: Sets the specified value as the ID of the OpenSearch index (`_id`).

delimiter()

Accepted values:	string
Default:	newline character

Description: By default, AxoSyslog separates the log messages of the batch with a newline character. You can specify a different delimiter by using the `delimiter()` option.

For details on how this option influences batch mode, see Batch mode and load balancing.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value

pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ## Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog created rule is there, packets can flow, otherwise the port is closed.

```
source {
    network(transport(udp)
    hook-commands(
          startup("iptables -I LOGCHAIN 1 -p udp --dport 514 -j ACCEPT")
          shutdown("iptables -D LOGCHAIN 1")
        )
     );
};
```

index()

Accepted values:	string or template
Default:	None

Description: The name of the OpenSearch index where OpenSearch will store the messages received from AxoSyslog. This option is mandatory for this destination.

You can use macros and template functions, but you must ensure that the resolved template contains only characters that OpenSearch permits in the name of the index. The AxoSyslog application does not validate the name of the index. For details on the characters permitted in the name of OpenSearch indices, see the documentation of OpenSearch.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

This destination supports only unencrypted key files (that is, the private key cannot be password-protected).

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

password()

Type:	string
Default:	

Description: The password that AxoSyslog uses to authenticate on the server where it sends the messages.

peer-verify()

Accepted values:	`yes` or `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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

> **Warning**
>
> **When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.**

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

proxy()

Type:	The proxy server address, in `proxy("PROXY_IP:PORT")` format. For example, `proxy("http://myproxy:3128")`
Default:	None

Description:

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

Alternatively, you can leave the HTTP as-is, in which case the driver leaves the default `http_proxy` and `https_proxy` environment variables unmodified.

> **Note**
>
> **Configuring the `proxy()` option overwrites the default `http_proxy` and `https_proxy` environment variables.**

Example: the proxy() option in configuration

The following example illustrates including the `proxy()` option in your configuration.

```
destination {
   http( url("SYSLOG_SERVER_IP:PORT") proxy("PROXY_IP:PORT") method("POST"));
 };
```

retries()

Type:	number (of attempts)

Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

To handle HTTP error responses, if the HTTP server returns 5xx codes, AxoSyslog will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, AxoSyslog will drop the messages.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

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

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

Declaration:

```
destination d_elasticsearch-http {
     elasticsearch-http(
         url("http://your-elasticsearch-server:9200/_bulk")
         type("")
         index("example-index")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

timeout()

Type:	number [seconds]
Default:	10

Description: The value (in seconds) to wait for an operation to complete, and attempt to reconnect the server if exceeded.

url()

Type:	URL or list of URLs, for example, url(“site1” “site2”)
Default:	N/A

Description: Specifies the hostname or IP address and optionally the port number of the OpenSearch indexer. Use a colon (`:`) after the address to specify the port number of the server. For example: `http://your-opensearch-indexer.server:8088/_bulk`

This option is mandatory for this destination.

Make sure that the URL ends with `_bulk`, this is the OpenSearch API endpoint that properly parses the messages sent by AxoSyslog.

In case the server on the specified URL returns a redirect request, AxoSyslog automatically follows maximum 3 redirects. Only HTTP and HTTPS based redirections are supported.

Starting with version 3.19, you can specify multiple URLs, for example, `url("site1" "site2")`. In this case, AxoSyslog sends log messages to the specified URLs in a load-balance fashion. This means that AxoSyslog sends each message to only one URL. For example, you can use this to send the messages to a set of ingestion nodes or indexers of your SIEM solution if a single node cannot handle the load. Note that the order of the messages as they arrive on the servers can differ from the order AxoSyslog has received them, so use load-balancing only if your server can use the timestamp from the messages. If the server uses the timestamp when it receives the messages, the order of the messages will be incorrect.

> **Warning**
>
> **If you set multiple URLs in the `url()` option, set the `persist-name()` option as well to avoid data loss.**

Starting with version AxoSyslog version 3.22, you can use any of the following formats to specify multiple URLs:

```
url("server1", "server2", "server3"); # comma-separated strings
url("server1" "server2" "server3"); # space-separated strings
url("server1 server2 server3"); # space-separated within a single string
```

user()

Type:	string
Default:	

Description: The username that AxoSyslog uses to authenticate on the server where it sends the messages.

use-system-cert-store()

Type:	`yes` or `no`
Default:	`no`

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation .

workers()

Type:	integer
Default:	4

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

Warning

`Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.30 osquery: Send 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 AxoSyslog to write to.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu. On RHEL and compatible distributions, the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"`, which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

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

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);
 };
```

8.30.1 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**

Default: N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type: string

Default: N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type: string

Default: N/A

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

`teardown()`

Type: string

Default: N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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.

8.31 Send logs, metrics, and traces to OpenTelemetry

AxoSyslog can send logs, metrics, and traces to OpenTelemetry over the OpenTelemetry Protocol (OTLP/gRPC) .

The only required parameter is the `url()` of the destination server, which includes the port number as well.

Prerequisites

- AxoSyslog version 4.3.0 or later.
- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Example: Forwarding OpenTelemetry data

```
log otel_forward_mode_alts {
  source {
    opentelemetry(
      port(12345)
      auth(alts())
    );
  };

  destination {
    opentelemetry(
      url("my-otel-server:12345")
      auth(alts())
    );
  };
};
```

Example: Sending log messages to OpenTelemetry

The following example receives syslog messages and forwards them as OpenTelemetry logs.

```
log non_otel_to_otel_tls {
  source {
    network(
      port(12346)
    );
  };

  destination {
    opentelemetry(
      url("my-otel-server:12346")
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
};
```

Options

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

```
opentelemetry(
  port(4317)
  auth(alts())
);
```

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_otlp {
    opentelemetry(
      url("your-otel-server:12346")
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog

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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

batch-bytes()

Accepted values:	number [bytes]
Default:	4MB

Available in AxoSyslog version 4.6 and later.

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option. The batch might be at most 1 message larger than the set limit.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

OTLP has a default 4 MiB batch limit, therefore the default value for `batch-bytes()` is 4 MB, which is a bit below 4 MiB.

The batch size is calculated before compression, which is the same as the limit is calculated on the server.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

channel-args()

Type:	arrow list
Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

compression()

Type:	boolean
Default:	`no`

Available in AxoSyslog version 4.5.0 and later.

Description: Enables compression in gRPC requests. Although gRPC supports various compression methods, currently only deflate is supported (which is basically the same as gzip).

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

Note

Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ### Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

headers()

Type:	arrow list
Default:	empty

Available in AxoSyslog 4.8 and later.

Description: Adds custom gRPC headers to each RPC call. Version 4.8 supported only static header names and values. For example:

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "axo-stream"
  )
```

Starting with version 4.9, you can use templates and macros in the header values.

```
headers(
    "organization" => "Axoflow"
    "stream-name" => "${HOST}"
  )
```

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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-alive()

Configures how AxoSyslog sends gRPC keepalive pings .

max-pings-without-data()

Type:	integer
Default:	

Description: The maximum number of gRPC pings that can be sent when there is no data/header frame to be sent. AxoSyslog won't send any pings after this limit. Set it to 0 disable this restriction and keep sending pings.

time()

Type:	number [milliseconds]
Default:	

Description: The period (in milliseconds) after which AxoSyslog sends a gRPC keepalive ping.

timeout()

Type:	number [milliseconds]
Default:	10

Description: The time (in milliseconds) AxoSyslog waits for an acknowledgement.

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property` : Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string` : Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message` : Drop the entire message silently, without logging the error.
- `silently-drop-property` : Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string` : Convert the property to string silently, without logging the error.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

response-action()

Type:	arrow list
Default:	Depends on the driver

Available in AxoSyslog version 4.11.0 and later.

Description: Fine-tunes how AxoSyslog behaves in case of different gRPC results. You can assign specific actions to the different gRPC results, for example:

```
response-action(
  not-found => disconnect
  unavailable => drop
)
```

The following gRPC results are supported:

- `aborted`
- `already-exists`
- `cancelled`
- `data-loss`
- `deadline-exceeded`
- `failed-precondition`
- `internal`
- `invalid-argument`
- `not-found`
- `ok`
- `out-of-range`
- `permission-denied`
- `resource-exhausted`
- `unauthenticated`
- `unavailable`

- `unknown`
- `unimplemented`

The following actions are available:

- `disconnect`
- `drop`
- `retry`
- `success`

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

send-time-zone()

Accepted values:	name of the timezone, or the timezone offset
Default:	local timezone

Description: Specifies the time zone associated with the messages sent by `syslog-ng`, if not specified otherwise in the message or in the destination driver. For details, see Timezones and daylight saving.

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

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

url()

Type:	string
Default:	`localhost:9095`

Description: The URL of the OpenTelemetry receiver.

worker-partition-autoscaling()

Type:	`yes`, `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)`, set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()`. Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()`.

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()`, `headers()`, or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()`.

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> ### Warning
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.32 pipe: Send 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);
```

> **Warning**
>
> **Starting with AxoSyslog 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"); };
```

8.32.1 pipe() destination options

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

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

check-program()

Type:	yes or no
Default:	no

Description:

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

create-dirs()

Type:	yes or no
Default:	no

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

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 (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

group()

Type:	string
Default:	Use the global settings

Description: Set the group of the created file to the one specified. To preserve the original properties of an existing file, use the option without specifying an attribute: `group()` .

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 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, AxoSyslog 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).

> **Warning**
>
> **Hazard of data loss! If the size of the incoming message is larger than the previously set `pad-size()` value, AxoSyslog 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), AxoSyslog 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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

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-reap()

Accepted values:	number (seconds)
Default:	60 or 0, see description for details

Description: The time to wait in seconds before an idle destination file or pipe is closed. Note that only destination files having macros in their filenames are closed automatically.

Starting with version 3.23, the way how `time-reap()` works is the following.

1. If the `time-reap()` option of the destination is set, that value is used, for example:

```
destination d_fifo {
    pipe(
        "/tmp/test.fifo",
        time-reap(30)  # sets time-reap() for this destination only
    );
};
```

2. If the `time-reap()` option of the destination is not set, and the destination does not use a template or macro in its filename or path, `time-reap()` is automatically set to 0. For example:

```
destination d_fifo {
    pipe(
        "/tmp/test.fifo",
    );
};
```

3. Otherwise, the value of the global `time-reap()` option is used, which defaults to 60 seconds.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

8.33 program: Send 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, AxoSyslog 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);
```

When using the `program()` driver, consider the following:

- The AxoSyslog 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 AxoSyslog to execute external applications.
- The AxoSyslog 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 AxoSyslog 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 AxoSyslog application stops, it will automatically stop the external program. To avoid restarting the application when AxoSyslog 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
```

8.33.1 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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()` .

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> ### Note
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> ### Warning
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> ### Note
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ## Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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` or `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, AxoSyslog). 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 AxoSyslog is reloaded.

mark-freq()

Accepted values:	number [seconds]
Default:	1200

Description: An alias for the obsolete `mark()` option, retained for compatibility with 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 3.4 and later.

Note that in earlier versions of AxoSyslog, 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), AxoSyslog 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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number

Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

8.34 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());
```

8.34.1 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**

Default: N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type: **string**

Default: N/A

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

`teardown()`

Type: **string**

Default: N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 AxoSyslog. 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.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

8.35 python: Write custom Python destinations

The Python destination allows you to write your own destination in Python. You can import external Python modules to process the messages, and send them to other services or servers. Since many services have a Python library, the Python destination makes integrating AxoSyslog very easy and quick.

The following points apply to using Python blocks in AxoSyslog in general:

- Python parsers and template functions are available in AxoSyslog version 3.10 and later.

  Python destinations and sources are available in AxoSyslog version 3.18 and later.

- Supported Python versions: 2.7 and 3.4+ (if you are using pre-built binaries, check the dependencies of the package to find out which Python version it was compiled with).

- The Python block must be a top-level block in the AxoSyslog configuration file.

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

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

- The Python object is initiated every time when AxoSyslog is started or reloaded.

  > **Warning**
  >
  > **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

- The Python block can contain multiple Python functions.

- Using Python code in AxoSyslog can significantly decrease the performance of AxoSyslog, especially if the Python code is slow. In general, the features of AxoSyslog are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The AxoSyslog application does not do any of this.
- Python error messages are available in the `internal()` source of AxoSyslog.
- You can access the name-value pairs of AxoSyslog directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of AxoSyslog. For details, see Logging from your Python code.

Note

Starting with 3.26, AxoSyslog assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your AxoSyslog configuration, add a unique `persist-name()` to each source or destination, otherwise AxoSyslog will not start. For example:

```
    log {
         source { python(class(PyNetworkSource) options("port" "8080") persist-
 name("<unique-string>); };
         source { python(class(PyNetworkSource) options("port" "8081")); };
     };
```

Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name` . For example:

```
    from syslogng import LogSource
       class PyNetworSource(LogSource):
         @staticmethod
         def generate_persist_name(options):
             return options["port"]
         def run(self):
             pass
         def request_exit(self):
             pass
```

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

Python destinations consist of two parts. The first is a AxoSyslog destination object that you define in your AxoSyslog configuration and use in the log path. This object references a Python class, which is the second part of the Python destination. The

Python class processes the log messages it receives, and can do virtually anything that you can code in Python. You can either embed the Python class into your AxoSyslog configuration file, or store it in an external Python file .

```
destination <name_of_the_python_destination>{
     python(
         class("<name_of_the_python_class_executed_by_the_destination>")
     );
 };

 python {
 class <name_of_the_python_class_executed_by_the_destination>(object):

     def open(self):
         """Open a connection to the target service

         Should return False if opening fails"""
         return True

     def close(self):
         """Close the connection to the target service"""
         pass

     def is_opened(self):
         """Check if the connection to the target is able to receive messages"""
         return True

     def init(self, options):
         """This method is called at initialization time

         Should return false if initialization fails"""
         return True

     def deinit(self):
         """This method is called at deinitialization time"""
         pass

     def send(self, msg):
         """Send a message to the target service

         It should return True to indicate success. False will suspend the
         destination for a period specified by the time-reopen() option."""
```

```
            return True

        def flush(self):
            """Flush the queued messages"""
            pass
    };
```

> **Note**
>
> **From version 3.27, AxoSyslog supports the arrow syntax in declaration. For more information, see the options() of the python() destination .**

Methods of the python() destination

init(self, options) method (optional)

The AxoSyslog application initializes Python objects every time when it is started or reloaded. The `init` method is executed as part of the initialization. You can perform any initialization steps that are necessary for your source to work.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

When this method returns with False, AxoSyslog does not start. It can be used to check options and return False when they prevent the successful start of the source.

`options` : This optional argument contains the contents of the `options()` parameter of the AxoSyslog configuration object as a Python dictionary.

is_opened(self) method (optional)

Checks if the connection to the target is able to receive messages, and should return True if it is. For details, see Error handling in the python() destination .

open(self) method (optional)

The `open(self)` method opens the resources required for the destination, for example, it initiates a connection to the target service. It is called after `init()` when AxoSyslog is started or reloaded. If `send()` returns with an error, AxoSyslog calls `close()` and `open()` before trying to send again.

If `open()` fails, it should return the False value. In this case, AxoSyslog retries it every `time-reopen()` seconds. By default, this is 1 second for Python sources and destinations, the value of `time-reopen()` is not inherited from the global option. For details, see Error handling in the python() destination .

send(self, message) method (mandatory)

The `send` method sends a message to the target service. It should return True to indicate success, or `self.QUEUED` when using batch mode. For other possible return values, see the description of the `flush()` method . Note that for batch mode, the `flush()` method must be implemented as well.

This is the only mandatory method of the destination.

If a message cannot be delivered after the number of times set in `retries()` (by default: 3), AxoSyslog drops the message and continues with the next message. For details, see Error handling in the python() destination .

The method can return True, False, or one of the following constants:

- `self.DROP` : The message is dropped immediately.
- `self.ERROR` : Corresponds to boolean False. The message is put back to the queue, and sending the message is attempted (up to the number of the `retries()` option). The destination is suspended for `time-reopen()` seconds.
- `self.SUCCESS` : Corresponds to boolean True. The message was sent successfully.
- `self.QUEUED` : The `send()` method should return this value when using batch mode, if it has successfully added the message to the batch. Message acknowledgment of batches is controlled by the `flush()` method.
- `self.NOT_CONNECTED` : The message is put back to the queue, and the destination is suspended. The `open()` method will be called, and the sending the messages will be continued with the same message/batch.

- `self.RETRY` : The message is put back to the queue, and sending the message is attempted (up to the number of the `retries()` option). If sending the message has failed `retries()` times, `self.NOT_CONNECTED` is returned.

flush(self) method (optional)

Send the messages in a batch. You can use this method to implement batch-mode message sending instead of sending messages one-by-one. When using batch mode, the `send()` method adds the messages to a batch (for example, a list), and the `flush()` method sends the messages as configured in the `batch-bytes()` , `batch-lines()` , or `batch-timeout()` options.

The method can return True, False, or one of the following constants:

- `self.DROP` : The messages cannot be sent and the entire batch is dropped immediately.
- `self.ERROR` : Corresponds to boolean False. The message is put back to the queue, and sending the message is attempted (up to the number of the `retries()` option). The destination is suspended for `time-reopen()` seconds.
- `self.SUCCESS` : Corresponds to boolean True. The message was sent successfully.
- `self.NOT_CONNECTED` : The message is put back to the queue, and the destination is suspended. The `open()` method will be called, and the sending the messages will be continued with the same message/batch.
- `self.RETRY` : The message is put back to the queue, and sending the message is attempted (up to the number of the `retries()` option). If sending the message has failed `retries()` times, `self.NOT_CONNECTED` is returned.

close(self) method (optional)

Close the connection to the target service. Usually it is called right before `deinit()` when stopping or reloading AxoSyslog. It is also called when `send()` fails.

The deinit(self) method (optional)

This method is executed when AxoSyslog is stopped or reloaded. This method does not return a value.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

Error handling in the python() destination

The Python destination handles errors as follows.

1. Currently AxoSyslog ignores every error from the `open` method until the first log message arrives to the Python destination. If the fist message has arrived and there was an error in the `open` method, AxoSyslog starts calling the `open` method every `time-reopen()` second, until opening the destination succeeds.
2. If the `open` method returns without error, AxoSyslog calls the `send` method to send the first message.
3. If the `send` method returns with an error, AxoSyslog calls the `is_opened` method.
    - If the `is_opened` method returns an error, AxoSyslog starts calling the `open` method every `time-reopen()` second, until opening the destination succeeds.
    - Otherwise, AxoSyslog calls the `send` method again.
4. If the `send` method has returned with an error `retries()` times and the `is_opened` method has not returned any errors, AxoSyslog drops the message and attempts to process the next message.

Example: Write logs into a file

The purpose of this example is only to demonstrate the basics of the Python destination, if you really want to write log messages into text files, use the file destination instead.

The following sample code writes the body of log messages into the `/tmp/example.txt` file. Only the `send()` method is implemented, meaning that AxoSyslog opens and closes the file for every message.

```
destination d_python_to_file {
     python(
         class("TextDestination")
     );
 };
 log {
     source(src);
     destination(d_python_to_file);
 };
 python {
 class TextDestination(object):
     def send(self, msg):
         self.outfile = open("/tmp/example.txt", "a")
         self.outfile.write("MESSAGE = %s\n" % msg["MESSAGE"])
         self.outfile.flush()
         self.outfile.close();
         return True
 };
```

The following code is similar to the previous example, but it opens and closes the file using the `open()` and `close()` methods.

```
destination d_python_to_file {
     python(
         class("TextDestination")
     );
 };
 log {
     source(src);
     destination(d_python_to_file);
 };
 python {
 class TextDestination(object):
     def open(self):
         try:
             self.outfile = open("/tmp/example.txt", "a")
             return True
         except:
             return False

     def send(self, msg):
         self.outfile.write("MESSAGE = %s\n" % msg["MESSAGE"])
         self.outfile.flush()
         return True

     def close(self):
         try:
             self.outfile.flush()
             self.outfile.close();
             return True
         except:
             return False
 };
```

Example: Print logs in batch mode

The following is a simple destination that uses the `flush()` method to print the messages in batch mode.

```
class MyDestination(object):
     def init(self, options):
         self.bulk = list()
         return True

     def send(self, msg):
          self.bulk.append(msg["MSG"].decode())
          return self.QUEUED

     def flush(self):
         print("flushing: " + ",".join(self.bulk))
         self.bulk = list()
         return self.SUCCESS
```

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

8.35.1 python() destination options

The Python destination allows you to write your own destination in Python. The `python()` destination has the following options. The `class()` option is mandatory. For details on writing destinations in Python, see python: Write custom Python destinations.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

This option does not have any effect unless the `flush()` method is implemented in the destination.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	25

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

This option does not have any effect unless the `flush()` method is implemented in the destination.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

This option does not have any effect unless the `flush()` method is implemented in the destination.

class()

Type:	string
Default:	N/A

Description: The name of the Python class that implements the destination, for example:

```
python(
     class("MyPythonDestination")
 );
```

If you want to store the Python code in an external Python file, the `class()` option must include the name of the Python file containing the class, without the path and the .py extension, for example:

```
python(
     class("MyPythonfilename.MyPythonDestination")
 );
```

For details, see Python code in external files

disk-buffer()

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

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	string
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing,

and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()`.

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

loaders()

Type:	list of python modules
Default:	empty list

Description: The AxoSyslog application imports Python modules specified in this option, before importing the code of the Python class. This option has effect only when the Python class is provided in an external Python file. This option has no effect when the Python class is provided within the AxoSyslog configuration file (in a `python{}` block). You can use the `loaders()` option to modify the import mechanism that imports Python class. For example, that way you can use hy in your Python class.

```
python(class(usermodule.HyParser) loaders(hy))
```

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

on-error()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property` : Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string` : Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message` : Drop the entire message silently, without logging the error.
- `silently-drop-property` : Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string` : Convert the property to string silently, without logging the error.

options()

Type:	string
Default:	N/A

Description: This option allows you to pass custom values from the configuration file to the Python code. Enclose both the option names and their values in double-quotes. The Python code will receive these values during initialization as the `options` dictionary. For example, you can use this to set the IP address of the server from the configuration file, so it is not hard-coded in the Python object.

```
python(
     class("MyPythonClass")
     options(
         "host" "127.0.0.1"
         "port" "1883"
         "otheroption" "value")
 );
```

For example, you can refer to the value of the host field in the Python code as `options["host"]`. Note that the Python code receives the values as strings, so you might have to cast them to the type required, for example: `int(options["port"])`

> **Note**
>
> From version 3.27, AxoSyslog supports the arrow syntax for declaring custom Java and Python options. You can alternatively declare them using a similar syntax:
>
> ```
> options(
>   "host" => "localhost"
>   "port" => "1883"
>   "otheroption" => "value"
> )
> ```

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")`.

> **Note**
>
> Starting with 3.26, AxoSyslog assigns a persist name to Python sources and destinations. The persist name is generated from the class name. If you want to use the same Python class multiple times in your AxoSyslog configuration, add a unique `persist-name()` to each source or destination, otherwise AxoSyslog will not start. For example:
>
> ```
> log {
>      source { python(class(PyNetworkSource) options("port" "8080") persist-
> name("<unique-string>); };
>      source { python(class(PyNetworkSource) options("port" "8081")); };
>  };
> ```
>
> Alternatively, you can include the following line in the Python package: `@staticmethod generate_persist_name`. For example:
>
> ```
> from syslogng import LogSource
>    class PyNetworSource(LogSource):
>      @staticmethod
>      def generate_persist_name(options):
>          return options["port"]
>      def run(self):
>          pass
>      def request_exit(self):
>          pass
> ```

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	1

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

value-pairs()

Type:	**parameter list of the `value-pairs()` option**
Default:	`scope("selected-macros" "nv-pairs")`

Description: The `value-pairs()` option creates structured name-value pairs from the data and metadata of the log message. For details on using `value-pairs()`, see Structuring macros, metadata, and other value-pairs.

> **Note**
>
> **Empty keys are not logged.**

You can use this option to limit which name-value pairs are passed to the Python code for each message. Note that if you use the `value-pairs()` option, the Python code receives the specified value-pairs as a Python dict. Otherwise, it receives the message object. In the following example, only the text of the log message is passed to Python.

```
destination d_python_to_file {
     python(
         class("pythonexample.TextDestination")
         value-pairs(key(MESSAGE))
     );
 };
```

8.36 redis: Store 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.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-redis` package on Debian/Ubuntu , or the `axosyslog-redis` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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: 4.29

    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);
    };
```

8.36.1 Batch mode and load balancing

Starting with version 3.34, you can send multiple log messages with the help of Redis's pipelining feature .

Batch size

The `batch-lines()` , `batch-lines()` , and `batch-timeout()` options of the destination determine how many log messages AxoSyslog sends in a batch. The `batch-lines()` option determines the maximum number of messages AxoSyslog puts in a batch in. This can be limited based on size and time:

AxoSyslog sends a batch every `batch-timeout()` milliseconds, even if the number of messages in the batch is less than `batch-lines()` . That way the destination receives every message in a timely manner even if suddenly there are no more messages.

To increase the performance of the destination, increase the number of worker threads for the destination using the `workers()` option, or adjust the `batch-lines()` and/or `batch-timeout()` options.

Example: Redis batch mode

The following destination sends log messages to a Redis server using the pipelining feature. A batch consists of `100` messages and is sent every `10` seconds (`10000` milliseconds) if there is less than `100` messages are in the queue.

```
destination d_redis {
     redis(
         host("localhost")
         port(6379)
         command("HINCRBY", "hosts", "$HOST", "1")
         batch-lines(100)
         batch-timeout(10000)
         log-fifo-size(100000)
     );
 };
```

8.36.2 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 AxoSyslog version 3.10 and later.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

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 AxoSyslog application ignores the return value of the command. If the Redis server returns an error, AxoSyslog 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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	string
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing,

and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()`.

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> ### Warning
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.37 riemann: Monitor 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.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-riemann` package on Debian/Ubuntu , or the `axosyslog-riemann` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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: 4.29

 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 AxoSyslog with the Riemann monitoring system, see the article A How to Guide on Modern Monitoring and Alerting by Fabien Wernli .

8.37.1 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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	string
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing,

and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()`.

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

event-time()

Type:	template, macro, or string
Default:	${UNIXTIME}

Description: Instead of the arrival time into Riemann, AxoSyslog 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, AxoSyslog defaults to the seconds version. In case the `event-time()` option is omitted altogether, AxoSyslog 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.

  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}")
    [...]
    );
 };
```

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

If an error occurs while sending the messages to the server, AxoSyslog 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), AxoSyslog 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()` , AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

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 AxoSyslog 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.

time-reopen()

Accepted values:	number [seconds]

Default:	60

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

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, AxoSyslog 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) AxoSyslog version 3.12.

cert-file()

Type:	path to a CA certificate in PEM format
Default:	

Description: Path to the a certificate file in PEM format. When establishing TLS connection, AxoSyslog 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 AxoSyslog 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, AxoSyslog 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 AxoSyslog version 3.7.

8.38 s3: Amazon S3

The `s3()` destination sends log messages to the Amazon Simple Storage Service (Amazon S3) object storage service. Messages are normally sent encrypted with TLS (HTTPS), but you can specify a custom unencrypted HTTP endpoint.

Prerequisites

- AxoSyslog version 4.4 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `s3()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in modules/python-modules/syslogng/modules/s3/scl/s3.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- An existing S3 bucket configured for programmatic access, and the related `ACCESS_KEY` and `SECRET_KEY` of a user that can access it. The user needs to have the following permissions:
  - `s3:ListBucket`
  - `s3:ListBucketMultipartUploads`
  - `s3:AbortMultipartUpload`
  - `s3:ListMultipartUploadParts`
  - `s3:PutObject`

  The following kms-related permissions are needed to use the `aws:kms` encryption. The AWS Role or User must have the following permissions on the given key:

  - `kms:Decrypt` (For details on why the `kms:Decrypt` is mandatory, check this AWS Knowledge Center entry .)
  - `kms:Encrypt`
  - `kms:GenerateDataKey`
- If you are not using the venv (`/usr/bin/syslog-ng-update-virtualenv`) created by AxoSyslog, you must install the `boto3` and/or `botocore` Python dependencies.

Declaration

```
s3(
    region("us-east-2")
    url("http://localhost:9000")
    bucket("syslog-ng")
    access-key("my-access-key")
    secret-key("my-secret-key")
    object-key("${HOST}/my-logs")
    template("${MESSAGE}\n")
);
```

Creating objects

AxoSyslog can create a new object based on the following strategies:

- Based on object size: The `max-object-size()` option configures AxoSyslog to finish an object if it reaches a certain size. AxoSyslog appends an index ("-1", “-2”, ...) to the end of the object key, then starts a new object.
- Based on timestamp: The `object-key-timestamp()` option can be used to set a datetime-related template, which is appended to the end of the object, for example: `"${R_MONTH_ABBREV}${R_DAY}"` . When a log message arrives with a newer timestamp template resolution, the previous timestamped object gets finished and a new one is started with the new timestamp. If an older message arrives, it doesn’t reopen the old object, but starts a new object with the key having an index appended to the old object.
- Based on timeout: The `flush-grace-period()` option sets the number of minutes to wait for new messages to arrive after the last one. If the timeout expires, AxoSyslog closes the object, and opens a new object (with an appended index) when a new message arrives.

All of these strategies can be used individually, or together.

The name of the object can be further modified by the following options:

- `object-key-suffix()` : A custom suffix that comes after the timestamp/index added by the object creation strategies.
- `compression()` : For compressed objects, `.gz` is appended to the very end of the object name.

To summarize, the different options (if set) modify the name of the object in the following order:

```
object-key()object-key-timestamp()max-object-size()object-key-suffix().gz(if compression is enabled)
```

Upload options

AxoSyslog uploads objects using the multipart upload API. AxoSyslog composes chunks locally. When a chunk reaches the size set in `chunk-size()` (by default 5 MiB), the chunk is uploaded. When an object is finished, the multipart upload is completed and S3 merges the chunks.

You can influence the upload via the `chunk-size()` , `upload-threads()` , and the `max-pending-uploads()` options.

Options

The following options are specific to the `s3()` destination.

access-key()

Type:	string
Default:	N/A

Description: The `ACCESS_KEY` of the service account used to access the S3 bucket. (Together with `secret-key()` .)

Starting with version 4.7, you can use the `AWS_...` environment variables or credentials files from the `~/.aws/` directory instead of this option. For details, see the official documentation .

bucket()

Type:	string
Default:	

Description: The name of the S3 bucket, for example, `my-bucket` . Note that the bucket must already exist.

canned-acl()

Type:	string
Default:	`empty`

Description: The ACL assigned to the object, if specified, for example, `bucket-owner-read` . The following values are valid:

`authenticated-read` , `aws-exec-read` , `bucket-owner-full-control` , `bucket-owner-read` , `log-delivery-write` , `private` , `public-read` , `public-read-write`

If you configure an invalid value, the default is used.

chunk-size()

Type:	string
Default:	`5MiB`

Description: The size of log messages that AxoSyslog writes to the S3 object in a batch. If compression is enabled, the `chunk-size()` refers to the compressed size.

compression()

Type:	boolean
Default:	`no`

Description: Setting `compression(yes)` enables gzip compression, and implicitly adds a `.gz` suffix to the very end of the created object's key. You can set the level of the compression using the `compresslevel()` option (0-9).

For a summary of how the different options affect the name of the object, see Creating objects .

compresslevel()

Type:	integer (0-9)
Default:	`9`

Description: Only has effect if `compression()` is set to yes. You can set the level of the compression using the `compresslevel()` option (0-9).

content-type()

Type:	string
Default:	`"application/octet-stream"`

Description: The content-type of the HTTP request.

flush-grace-period()

Type:	integer [minutes]
Default:	`60`

Description: After the grace period expires and no new messages are routed to the destination, AxoSyslog flushes the contents of the buffer to the S3 object even if the volume of the messages in the buffer is lower than `chunk-size()` .

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

kms-key()

Type:	string
Default:	N/A

Available in AxoSyslog 4.8 and later.

Description: The `kms-key()` used for server-side encryption . The value of the `kms-key()` parameter must be one of the following:

- The ID of a key.
- An alias of a key. In that case, make sure to add the alias/prefix, for example: `kms-key("alias/log-archive")`
- The ARN of a key.

max-object-size()

Type:	string
Default:	`5120GiB`

Description: The maximal size of the S3 object. If an object reaches this size, AxoSyslog appends an index ("-1", “-2”, ...) to the end of the object key and starts a new object after rotation.

For a summary of how the different options affect the name of the object, see Creating objects .

max-pending-uploads()

Type:	integer
Default:	`32`

Description: The `max-pending-uploads()` and `upload-threads()` options configure the upload of the chunks. Uploading happens in multiple threads to minimize network overhead.

- `upload-threads()` limits the maximum number of parallel uploads.
- `max-pending-uploads()` limits the number of chunks that are waiting in the work queue of the upload threads to get uploaded.

object-key()

Type:	template
Default:	N/A

Description: The object key (or key name), which uniquely identifies the object in an Amazon S3 bucket. Note that a suffix may be appended to this object key depending on the naming strategies and other options used. Example: `my-logs/${HOSTNAME}/` .

object-key-suffix()

Type:	template
Default:	empty string

Available in AxoSyslog 4.20 and later.

Description: A suffix added to the object key.

For a summary of how the different options affect the name of the object, see Creating objects .

object-key-timestamp()

Type:	template
Default:	

Description: The `object-key-timestamp()` option can be used to set a datetime-related template, which is appended to the end of the object key, for example: `"${R_MONTH_ABBREV}${R_DAY}"` . When a log message arrives with a newer timestamp template resolution, the previous timestamped object gets finished and a new one is started with the new timestamp. If an older message arrives, it doesn't reopen the old object, but starts a new object with the key having an index appended to the old object.

For a summary of how the different options affect the name of the object, see Creating objects .

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

region()

Type:	string
Default:	

Description: The AWS region to use when writing the bucket. This should normally be the same region where the bucket is created. This option implies an API endpoint `url()` . For providers other than AWS, or for custom API endpoints, use the `url()` option.

role()

Type:	string
Default:	

Description: Assume the specified role when accessing S3.

secret-key()

Type:	string
Default:	N/A

Description: The `SECRET_KEY` of the service account used to access the S3 bucket. (Together with `access-key()` .)

Starting with version 4.7, you can use the `AWS_...` environment variables or credentials files from the `~/.aws/` directory instead of this option. For details, see the official documentation .

server-side-encryption()

Type:	string
Default:	N/A

Available in AxoSyslog 4.8 and later.

Description: You can use the `server-side-encryption()` and `kms-key()` options to configure encryption. Currently only `server-side-encryption("aws:kms")` is supported.

```
destination d_s3 {
  s3(
    bucket("log-archive-bucket")
    object-key("logs/syslog")
    server-side-encryption("aws:kms")
    kms-key("alias/log-archive")
  );
```

For details on using KMS keys, see the official S3 documentation .

storage-class()

Type:	string
Default:	`STANDARD`

Description: The storage class of the object , for example, `REDUCED_REDUNDANCY` . The following values are valid:

`DEEP_ARCHIVE` , `GLACIER` , `GLACIER_IR` , `INTELLIGENT_TIERING` , `ONEZONE_IA` , `OUTPOSTS` , `REDUCED_REDUNDANCY` , `SNOW` , `STANDARD` , `STANDARD_IA`

If you configure an invalid value, the default is used.

upload-threads()

Type:	integer
Default:	`8`

Description: The number of AxoSyslog worker threads that are used to upload data to S3 from this destination.

template()

Type:	template or template function
Default:	`${MESSAGE}\n`

Description: The message as written to the Amazon S3 object. You can use templates and template functions to format the message.

url()

Type:	string
Default:	N/A

Description: The API endpoint URL for writing to the S3 bucket, for example `https://s3.us-west-2.amazonaws.com`, `http://minio.local:9000`, or `https://storage.googleapis.com`.

8.39 slack: Send alerts and notifications to a Slack channel

The `slack()` destination driver sends messages to a Slack channel using the Slack Web API. For the list of available optional parameters, see Slack destination options.

Prerequisites

- AxoSyslog version 3.19 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `slack()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/slack/slack.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- Create a Slack app and a Webhook that AxoSyslog can use. For details, see the Slack documentation .

Declaration:

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

destination d_slack {
  slack(
    hook-
url("https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
)
  );
};
```

The driver allows you to modify nearly every field of the HTTP request. For details, see the Slack API documentation .

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

By default, the `throttle()` option is set to 1, because Slack has a 1 message/second limit on Webhooks. It can allow more message in short bursts, so you can set it to 0, if you only expect messages in a short period of time. For details, see the Web API rate limiting in the Slack documentation .

Example: Using the slack() driver

The following example sets the colors and the author of the message.

```
@include "scl.conf"

destination d_slack1 {
  slack(
    hook-
url("https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX"
)
    colors("#000000,#222222,#444444,#666666,#888888,#AAAAAA,#CCCCCC,#EEEEEE")
    color-chooser(7)
    author-name("Example BOT")
    author-link("https://www.syslog-ng.com/products/open-source-log-management")
    author-icon("https://raw.githubusercontent.com/MrAnno/vscode-syslog-
ng/master/images/syslog-ng-icon.png")
  );
};
```

8.39.1 Slack destination options

The `slack` destination of AxoSyslog can directly post log messages and notifications to Slack channels. The `slack` destination has the following options.

author-name()

Type:	string or template
Default:	'host: ${HOST} | program: ${PROGRAM}(${PID}) | severity: ${PRIORITY}'

Description: The sender of the message as displayed in Slack. For details, see the author_name option in the Slack documentation .

author-link()

Type:	string or URL
Default:	None

Description: A hyperlink for the sender of the message as displayed in Slack. For details, see the author_link option in the Slack documentation .

author-icon()

Type:	URL
Default:	None

Description: A hyperlink for icon of the author to be displayed in Slack. For details, see the author_icon option in the Slack documentation .

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

For details on how this option influences batch mode, see http: Posting messages over HTTP without Java

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

For details on how this option influences batch mode, see http: Posting messages over HTTP without Java

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()` , `ca-file()` , `cert-file()` , `cipher-suite()` , `key-file()` , `peer-verify()` , and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()` . Avoid mixing the two methods. 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 AxoSyslog application uses this certificate to validate the certificate of the HTTPS server. If the file contains a certificate chain, the file must begin with the certificate of the host, followed by the CA certificate that signed the certificate of the host, and any other signing CAs in order.

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()` , `ca-file()` , `cert-file()` , `cipher-suite()` , `key-file()` , `peer-verify()` , and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()` . Avoid mixing the two methods. 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 AxoSyslog 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 AxoSyslog. 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 AxoSyslog receives TLS-encrypted connections, the order of ciphers set on the AxoSyslog server takes precedence over the client settings.

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

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

Declaration:

```
destination d_http {
     http(
         url("http://127.0.0.1:8080")
         tls(
             ca-dir("dir")
             ca-file("ca")
             cert-file("cert")
             cipher-suite("cipher")
             key-file("key")
             peer-verify(yes|no)
             ssl-version(<the permitted SSL/TLS version>)
         )
     );
 };
```

colors()

Type:	list of colors in hexadecimal format
Default:	'#512E5F,#B03A2E,#E74C3C,#F39C12,#F8C471,#7DCEA0,#5DADE2,#85929E'

Description: The colors to be assigned to the messages of different importance levels.

color-chooser()

Type:	integer or template
Default:	'${LEVEL_NUM}'

Description: An integer that assigns a color to the message from the list of colors set in the `colors()` option.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

Note

Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ### Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

fallback()

Type:	string or template
Default:	'${MSG} - host: ${HOST}	program: ${PROGRAM}(${PID})	severity: ${PRIORITY}'

Description: The plain-text summary of the Slack attachment. For details, see the fallback option in the Slack documentation .

footer()

Type:	URL
Default:	string or template

Description: The footer of the message. For details, see the footer option in the Slack documentation .

footer-icon()

Type:	URL
Default:	None

Description: A hyperlink for an image. For details, see the footer_icon option in the Slack documentation .

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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")
        )
     );
};
```

hook-url()

Type:	URL
Default:	None

Description: The Webhook URL for the Incoming Webhook of your Slack app. This URL must also include the authentication token that AxoSyslog uses to authenticate to Slack. For example:
`https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX`

For details, see the Slack documentation about Incoming Webhooks .

image-url()

Type:	URL
Default:	None

Description: A hyperlink for an image. For details, see the image_url option in the Slack documentation .

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

pretext()

Type:	string or template
Default:	None

Description: The text that appears above the attachment block. For details, see the pretext option in the Slack documentation .

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

To handle HTTP error responses, if the HTTP server returns 5xx codes, AxoSyslog will attempt to resend messages until the number of attempts reaches `retries`. If the HTTP server returns 4xx codes, AxoSyslog will drop the messages.

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

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

Make sure that you specify TLS options either using their own dedicated option (`ca-dir()`, `ca-file()`, `cert-file()`, `cipher-suite()`, `key-file()`, `peer-verify()`, and `ssl-version()`), or using the `tls()` block and inserting the relevant options within `tls()`. Avoid mixing the two methods. 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 AxoSyslog. 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.

By default, the `throttle()` option is set to 1, because Slack has a 1 message/second limit on Webhooks. It can allow more message in short bursts, so you can set it to 0, if you only expect messages in a short period of time. For details, see the Web API rate limiting in the Slack documentation .

thumb-url()

Type:	URL
Default:	None

Description: A hyperlink for a thumbnail image. For details, see the thumb_url option in the Slack documentation .

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.

title()

Type:	string or template
Default:	None

Description: The message title in Slack. For details, see the title option in the Slack documentation .

title-link()

Type:	URL
Default:	None

Description: A hyperlink for the message title in Slack. For details, see the title_link option in the Slack documentation .

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.

use-system-cert-store()

Type:	`yes` or `no`
Default:	`no`

Description: Use the certificate store of the system for verifying HTTPS certificates. For details, see the curl documentation .

workers()

Type:	integer

Default: 1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

> **Warning**
>
> `Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

If you are using load-balancing (that is, you have configured multiple servers in the `url()` option), increase the number of worker threads at least to the number of servers. For example, if you have set three URLs (`url("site1", "site2", "site3")`), set the `workers()` option to 3 or more.

8.40 smtp: Generate SMTP messages (emails) from logs

The destination is aimed at a fully controlled local, or near-local, trusted SMTP server. The goal is to send mail to trusted recipients, through a controlled channel. It hands mails over to an SMTP server, and that is all it does, therefore the resulting solution is as reliable as sending an email in general. For example, AxoSyslog does not verify whether the recipient exists.

The `smtp()` driver sends email messages triggered by log messages. The `smtp()` driver uses SMTP, without needing external applications. You can customize the main fields of the email, add extra headers, send the email to multiple recipients, and so on.

The `subject()`, `body()`, and `header()` fields may include macros which get expanded in the email. For more information on available macros see Macros of AxoSyslog.

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.

Prerequisites

- AxoSyslog version 3.4 or later.
- This feature requires a separate module. Install the `axosyslog-mod-smtp` package on Debian/Ubuntu , or the `axosyslog-smtp` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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("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 alerting service detected the following important log
message:\n $MSG\n-- \n")
        );
    };
```

The following example sets some optional parameters as well.

```
   destination d_smtp {
        smtp(
            host("localhost")
            port(25)
            from("syslog-ng alert service" "noreply@example.com")
            to("Admin #1" "admin1@example.com")
            to("Admin #2" "admin2@example.com")
            cc("Admin BOSS" "admin.boss@example.com")
            bcc("Blind CC" "blindcc@example.com")
            subject("[ALERT] Important log message of $LEVEL condition received
from $HOST/$PROGRAM!")
            body("Hi!\nThe syslog-ng alerting service detected the following
important log message:\n $MSG\n-- \nsyslog-ng\n")
            header("X-Program", "$PROGRAM")
            );
    };
```

Example: Simple email alerting with the smtp() driver

The following example sends an email alert if the eth0 network interface of the host is down.

```
filter f_linkdown {
     match("eth0: link down" value("MESSAGE"));
 };
 destination d_alert {
     smtp(
         host("localhost") port(25)
         from("syslog-ng alert service" "syslog@localhost")
         reply-to("Admins" "root@localhost")
         to("Ennekem" "me@localhost")
         subject("[SYSLOG ALERT]: eth0 link down")
         body("Syslog received an alert:\n$MSG")
         );
 };

 log {
     source(s_local);
     filter(f_linkdown);
     destination(d_alert);
 };
```

8.40.1 smtp() destination options

The `smtp()` sends email messages using SMTP, without needing external applications. The `smtp()` destination has the following options:

body()

Type:	string
Default:	n/a

Description: The BODY field of the email. You can also use macros in the string. Use `\\n` to start a new line. For example:

```
body("AxoSyslog received the following alert from $HOST:\n$MSG")
```

bcc()

Type:	string
Default:	n/a

Description: The BCC recipient of the email (contents of the BCC field). You can specify the email address, or the name and the email address. Set the `bcc` option multiple times to send the email to multiple recipients. For example: `bcc("admin@example.com")` or `bcc("Admin" "admin@example.com")` or `>bcc("Admin" "admin@example.com") bcc("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

cc()

Type:	string
Default:	n/a

Description: The CC recipient of the email (contents of the CC field). You can specify the email address, or the name and the email address. Set the `cc` option multiple times to send the email to multiple recipients. For example: `cc("admin@example.com")` or `cc("Admin" "admin@example.com")` or `cc("Admin" "admin@example.com") cc("Admin2" "admin2@example.com"`

You can also use macros to set the value of this parameter.

disk-buffer()

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

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ## Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

from()

Type:	string
Default:	n/a

Description: The sender of the email (contents of the FROM field). You can specify the email address, or the name and the email address. For example:

```
from("admin@example.com")
```

or

```
from("Admin" "admin@example.com")
```

If you specify the `from()` option multiple times, the last value will be used. Instead of the `from()` option, you can also use `sender()`, which is just an alias of the `from()` option.

You can also use macros to set the value of this parameter.

header()

Type:	string
Default:	n/a

Description: Adds an extra header to the email with the specified name and content. The first parameter sets the name of the header, the second one its value. The value of the header can contain macros. Set the `header` option multiple times to add multiple headers. For example:

```
header("X-Program", "$PROGRAM")
```

When using the header option, note the following points:

- Do not use the `header()` option to set the values of headers that have dedicated options. Use it only to add extra headers.
- If you set the same custom header multiple times, only the first will be added to the email, other occurrences will be ignored.
- It is not possible to set the DATE, Return-Path, Original-Recipient, Content-*, MIME-*, Resent-*, Received headers.

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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"`, AxoSyslog will use a socket to connect to the local SMTP server. Use `host="127.0.0.1"` to force TCP communication between AxoSyslog and the local SMTP server.**

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

port()

Type:	number
Default:	25

Description: The port number of the SMTP server.

reply-to()

Type:	string
Default:	n/a

Description: Replies of the recipient will be sent to this address (contents of the REPLY-TO field). You can specify the email address, or the name and the email address. Set the `reply-to()` option multiple times to send the email to multiple recipients. For example: `reply-to("admin@example.com")` or `reply-to("Admin" "admin@example.com")` or `reply-to("Admin" "admin@example.com") reply-to("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

retries()

Type:	number (of attempts)
Default:	3

Description: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()` .

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

subject()

Type:	string
Default:	n/a

Description: The SUBJECT field of the email. You can also use macros. For example:

```
subject("[SYSLOG ALERT]: Critical error message received from $HOST")
```

If you specify the `subject()` option multiple times, the last value will be used.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

to()

Type:	string
Default:	localhost

Description: The recipient of the email (contents of the TO field). You can specify the email address, or the name and the email address. Set the `to()` option multiple times to send the email to multiple recipients. For example: `to("admin@example.com")` or `to("Admin" "admin@example.com")` or `to("Admin" "admin@example.com") to("Admin2" "admin2@example.com")`

You can also use macros to set the value of this parameter.

8.41 snmp: Send SNMP traps

The `snmp()` driver sends SNMP traps using the Simple Network Management Protocol version 2c or version 3. Incoming log messages can be converted to SNMP traps, as the fields of the SNMP messages can be customized using AxoSyslog macros.

The `snmp()` driver is available in AxoSyslog version 3.22 and later.

> **Note**
>
> **The `snmp` destination driver currently supports sending SNMP traps only using the UDP transport protocol.**

The `snmp()` driver requires the `host()`, `trap-obj()`, and `snmp-obj()` options to be set, as well as the `engine-id()` and `version()` options when using the SNMPv3 protocol. For the list of available optional parameters, see snmp() destination options.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-snmp` package on Debian/Ubuntu, or the `axosyslog-afsnmp` package on RHEL and compatible distributions. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

Declaration:

```
destination d_snmp {snmp(host() trap-obj() snmp-obj() ...);};
```

> **Warning**
>
> **If AxoSyslog cannot resolve the destination hostname during startup, it will try to resolve the hostname again when the next message to be sent as an SNMP trap is received. However, if this name resolution fails, the trap will be dropped.**

> **Note**
>
> **The `snmp()` destination driver does not generate MARK signals itself, but can receive and forward MARK signals.**

Example: Using the snmp() destination driver

The following example defines an SNMP destination that uses the SNMPv2c protocol.

```
   destination d_snmpv2c{
        snmp(
            version('v2c')
            host('192.168.1.1')
            trap-obj('.1.3.6.1.6.3.1.1.4.1.0', 'Objectid',
'.1.3.6.1.4.1.18372.3.1.1.1.2.1')
            snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.1.0', 'Octetstring', 'Test SNMP
trap')
            snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.2.0', 'Octetstring', 'admin')
            snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.3.0', 'Ipaddress',
'192.168.1.1')
            );
    };
```

The following example defines an SNMP destination that uses the SNMPv3 protocol and uses macros to fill the values of the SNMP objects.

```
   destination d_snmpv3{
        snmp(
            version('v3')
            host('192.168.1.1')
            port(162)
            engine-id('0xdeadbeefde')
            auth-username('myusername')
            auth-password('password')
            enc-algorithm('AES')
            enc-password('password')
            trap-obj('.1.3.6.1.6.3.1.1.4.1.0', 'Objectid',
'.1.3.6.1.4.1.18372.3.1.1.1.2.1')
            snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.1', 'Octetstring', '${MESSAGE}')
            snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.2', 'Octetstring', 'admin')
            snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.3', 'Ipaddress', '${SOURCEIP}')
            );
    };
```

8.41.1 snmp() destination options

This driver sends SNMP traps using the SNMP v2c or v3 protocol.

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

auth-algorithm()

Type:	`SHA` or `sha`
Default:	SHA

Description: The authentication method to use. Lowercase values (for example, `sha`) can be used as well.

This option is used with the SNMPv3 protocol.

auth-password()

Type:	string
Default:	empty string

Description: The password used for authentication. If the `auth-username()` option is set but the `auth-password()` is empty, AxoSyslog will try to authenticate with an empty password.

This option is used with the SNMPv3 protocol.

auth-username()

Type:	string
Default:	empty string

Description: The username used to authenticate on the SNMP server. If this parameter is set, AxoSyslog will try to authenticate on the SNMP server.

This option is used with the SNMPv3 protocol.

community()

Type:	string
Default:	public

Description: The community string used for SNMPv2c authentication.

This option is used with the SNMPv2c protocol.

enc-algorithm()

Type:	`AES` or `aes`
Default:	AES

Description: The encryption method used to encrypt the SNMP traffic. Lowercase values (for example, `aes`) can be used as well.

This option is used with the SNMPv3 protocol.

enc-password()

Type:	string
Default:	

Description: The password used for the encryption. Encryption is used only if the `enc-password()` is not empty.

This option is used with the SNMPv3 protocol.

engine-id()

Type:	number (hexadecimal number)
Default:	

Description: The engine ID is a hexadecimal number at least 10 digits long, starting with `0x` . for example, `0xABABABABAB` .

This option is a required parameter when using the SNMPv3 protocol.

host()

Type:	hostname or IP address
Default:	n/a

Description: Hostname of the SNMP server.

log-fifo-size()

Type:	number
Default:	Use global setting.

Description: The number of messages that the output queue can store.

port()

Type:	number (port number)
Default:	162

Description: The port number to connect to.

snmp-obj()

Type:	<oid_of_the_object> <type_of_the_object»<value_of_the_object> >
Default:	n/a

Description: The `snmp-obj()` option can be used to create custom SNMP trap elements. To create a trap element, specify the OID, type, and value of the element in the `snmp-obj()` option. To send SNMP traps, at least one `snmp-obj()` option must be defined. The `snmp-obj()` option requires the following parameters. Note that AxoSyslog does not validate the values of these elements.

- `<oid_of_the_object>: The object id of the SNMP object, for example,` .1.3.6.1.4.1.18372.3.1.1.1.1.1`.
- `<type_of_the_object>: The type of the object specified as an ASN.1 primitive. One of:` Integer, Timeticks, Octetstring, Counter32, Ipaddress, Objectid`. The type names are not case sensitive.
- `<value_of_the_object>: The value of the object as a string. The macros of AxoSyslog can be used to set these values, making it possible to transfer the content and other metadata from the the syslog message to the SNMP trap. Note that if the value of an` Integer, Counter32 `or` Timeticks` object is not a number (for example, is an empty string or other not-number string), AxoSyslog will automatically replace the value with 0. The values of other types of objects are not validated.

Example: Defining SNMP objects

The following are SNMP object definitions:

```
snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.3', 'Ipaddress', '192.168.1.1')
```

```
snmp-obj('.1.3.6.1.4.1.18372.3.1.1.1.1.2', 'Octetstring', '${MESSAGE}')
```

time-zone()

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

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

The timezone can be specified 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.

trap-obj()

Type:	<oid_of_the_object> “Objectid”, <value_of_the_object»
Default:	n/a

Description: The `trap-obj()` is a specialized version of the `snmp-obj()` option that is used to identify the SNMP trap object. The type of the trap object is always `Objectid`. The `<oid_of_the_object> and the` <value_of_the_object»parameters are identical to the respective parameters of the `snmp-obj()` option. For details on these parameters, see snmp-obj() .

> **Note**
>
> **Using the `trap-obj()` object is equivalent to using the `snmp-obj()` with the `Objectid` type.**

version()

Type:	`v2c` or `v3`
Default:	v2c

Description: Specifies which version of the SNMP protocol to use.

> **Note**
>
> **The AxoSyslog application will accept any valid option for the `snmp()` destination, but will only use the ones relevant to the selected protocol version, any other option will be ignored. For example, if the `version("v2c") engine-id("0xABABABABAB") community("mycommunity")` options are set, AxoSyslog will accept every option, but process only the `community()` option, because `engine-id()` applies only to SNMPv3.**

8.42 splunk-hec-event: Send messages to Splunk HEC

AxoSyslog can send messages to the Splunk HTTP Event Collector (HEC).

Prerequisites

- AxoSyslog version 4.2.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- Enable the HTTP Event Collector (HEC) on your Splunk deployment.
- Create a token for AxoSyslog to use in the `token()` option of the destination. When creating the token, use the syslog source type.

  For details, see Set up and use HTTP Event Collector in Splunk Web .

HEC events API

The `splunk-hec-event()` destination feeds Splunk via the HEC events API .

Minimal configuration:

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

destination d_splunk_hec_event {
  splunk-hec-event(
    url("https://localhost:8088")
    token("70b6ae71-76b3-4c38-9597-0c5b37ad9630")
  );
};
```

Additional options include:

```
event()
index()
source()
sourcetype()
host()
time()
default-index()
default-source()
default-sourcetype()
fields()
extra-headers()
extra-queries()
content-type()
```

`event()` accepts a template, which declares the content of the log message sent to Splunk. Default value: `${MSG}`

`index()`, `source()`, `host()`, and `time()` accept templates, and declare the respective field of each log message based on the set template.

`default-index()`, `default-source()`, and `default-sourcetype()` accept literal strings, and are used as fallback values if a log message doesn't set these fields. These values are passed to the URL as query parameters, so they don't inflate the body of the HTTP request for each message in the batch, which saves bandwidth.

`fields()` accepts template, which is passed as additional indexing metadata to Splunk.

`extra-headers()` , `extra-queries()` , and `content-type()` are additional HTTP request options.

HEC raw API

The `splunk-hec-raw()` destination feeds Splunk via the HEC raw API .

Minimal configuration:

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

destination d_splunk_hec_raw {
  splunk-hec-raw(
    url("https://localhost:8088")
    token("70b6ae71-76b3-4c38-9597-0c5b37ad9630")
    channel("05ed4617-f186-4ccd-b4e7-08847094c8fd")
  );
};
```

The options of the `splunk-hec-raw()` destination are similar to the `splunk-hec-event()` destination, but it has a mandatory option: `channel()` . The `channel()` option must be a globally unique channel identifier (GUID) , this ID differentiates the data from different clients. Note that Splunk doesn't generate this ID, you must create it for yourself. When Splunk sees a new channel identifier, it creates a new channel.

Use the `template()` option to set the content of the log message sent to Splunk (and not the `event()` option that is used in the `splunk-hec-event()` destination).

8.43 sql: Store 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. Starting with AxoSyslog 4.0, type information is automatically added to the stored columns if available. For details, see Specifying data types in value-pairs.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-sql` package on Debian/Ubuntu , or the `axosyslog-sql` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

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

The `sql()` driver has the following required parameters: `type()` , `database()` , `table()` , `columns()` , and `values()` .

> **Warning**
>
> The AxoSyslog application requires read and write access to the SQL table, otherwise it cannot verify that the destination table exists.
>
> Currently the AxoSyslog 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.

> **Note**
>
> **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 AxoSyslog).

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

Inserting the records into the database is performed by a separate thread. The AxoSyslog 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 AxoSyslog 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"));
    };
```

8.43.1 Using the sql() driver with an Oracle database

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

- The hostname of the database server is set in the `tnsnames.ora` file, not in the `host` parameter of the `sql()` destination.

  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 AxoSyslog 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 AxoSyslog.

- 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 AxoSyslog, 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 AxoSyslog 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 AxoSyslog 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)
    )
    )
```

8.43.2 Using the sql() driver with a Microsoft SQL database

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

- The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the AxoSyslog 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 AxoSyslog, the types of database columns must be explicitly set for the MSSQL destination.

> **Warning**
>
> **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.

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 AxoSyslog 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 AxoSyslog server. Edit or create this file as needed for your configuration. A sample is provided below.

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

8.43.3 Interacting with the database

SQL operations

Create table:

- If the given table does not exist, AxoSyslog tries to create it with the given column types.
- The AxoSyslog application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.
- If AxoSyslog 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, AxoSyslog tries to add columns in this table but never will delete or modify an existing column.
- If AxoSyslog cannot create or alter a table, it tries to do it again when reach the next `time-reopen()`.
- The AxoSyslog 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 AxoSyslog application automatically performs the escaping required to insert the messages into the database.
- If insert returns with error, AxoSyslog tries to insert the message +two times by default, then drops it. Retrying time is the value of `time-reopen()`.

Encoding

The AxoSyslog application uses UTF-8 by default when writes logs into database.

Start/stop and reload

Start:

- The AxoSyslog application will connect to database automatically after starting regardless existing incoming messages.

Stop:

- The AxoSyslog application will close the connection to database before shutting down.

Possibility of losing logs:

- The AxoSyslog application cannot lose logs during shutting down if disk buffer was given and it is not full yet.
- The AxoSyslog application cannot lose logs during shutting down if disk buffer was not given.

Reload:

- The AxoSyslog 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 overridden by sql driver regardless of local or relayed incoming message.

It will be grown continuously.

8.43.3.1 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`.

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

8.43.4 sql() destination options

This driver sends messages into an SQL database. The `sql()` destination has the following options:

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

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, AxoSyslog creates `text` columns. Note that not every database engine can index text fields.

> **Warning**
>
> **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 AxoSyslog 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 AxoSyslog 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)
```

dbdi-driver-dir()

Type:	string
Default:	empty string

Description: Defines an optional DBI driver location for DBD initialization.

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:

capacity-bytes()

Type:	number (bytes)
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

Note

Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> ### Warning
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 AxoSyslog from creating non-existing database tables automatically. The AxoSyslog application typically has to create tables if you use macros in the table names. Available in AxoSyslog version 3.2 and later.

- *explicit-commits*: By default, AxoSyslog 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 `batch-lines()` parameter. The `explicit-commits` option is available in AxoSyslog 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)
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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"`, AxoSyslog will use a socket to connect to the local database server. Use `host="127.0.0.1"` to force TCP communication between AxoSyslog 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 AxoSyslog to request indexing on the default columns.

The AxoSyslog 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.

quote-char()

Type:	string

Default:	

Available in AxoSyslog version 4.3.0 and newer.

Description: Set custom quoting for table and index names (for example, MySQL needs sometimes this for certain identifiers).

> Note: Using a backtick character needs special formatting, because AxoSyslog uses backticks for configuration parameter names. To use backticks as quote character, set a double backtick: `quote-char("``")`

retries()

Type:	number (insertion attempts)
Default:	3

Description: The number of insertion attempts. If AxoSyslog 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, AxoSyslog 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 AxoSyslog has successfully connected to the database. For example:

```
    session-statements("SET COLLATION_CONNECTION='utf8_general_ci'")
```

> **Warning**
>
> **The AxoSyslog 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-reopen()

Accepted values:	number [seconds]
Default:	60

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

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)
```

8.44 stdout: Send messages to standard output

Available in AxoSyslog version 4.4 and later.

The `stdout()` destination driver sends messages to the standard output.

Declaration:

```
log {
    source{ stdin(); };
    destination{ stdout(); };
};
```

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 (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

flush-timeout() (DEPRECATED)

Type:	time in milliseconds
Default:	10000 [milliseconds]

This option is deprecated. Several destination drivers support the `batch-timeout()` option instead, which you can set at the destination level.

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends flushes to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `flush-timeout()` seconds.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the

original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

local-time-zone()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Sets the timezone used when expanding filename and tablename templates.

The timezone can be specified 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 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 3.4 and later.

pad-size()

Type:	number
Default:	0

Description: If set, AxoSyslog 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).

> **Warning**
>
> **Hazard of data loss! If the size of the incoming message is larger than the previously set `pad-size()` value, AxoSyslog will truncate the message to the specified size. Therefore, all message content above that size will be lost.**

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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")` .

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.

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), AxoSyslog 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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

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.**

8.45 stomp: Publish 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. AxoSyslog supports version 1.0 of the STOMP protocol. The AxoSyslog `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.

Prerequisites

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-stomp` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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))
     );
 };
```

8.45.1 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` or `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, AxoSyslog waits until the server confirms that it has received the message. This delay can seriously limit the performance of AxoSyslog 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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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` or `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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

8.46 Sumo Logic destinations: sumologic-http() and sumologic-syslog()

AxoSyslog can send log messages to Sumo Logic , a cloud-based log management and security analytics service, by using the `sumologic-http()` and `sumologic-syslog()` destinations.

Prerequisites

Currently, using the `sumologic-http()` and `sumologic-syslog()` destinations with AxoSyslog has the following prerequisites:

- AxoSyslog version 3.27.1 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `sumologic-http()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/sumologic/sumologic.conf on GitHub.
- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .
- A Sumo Logic account.

  If you do not yet have a Sumo Logic account, visit the official Sumo Logic website , and click `Start free trial` to create an account.

  > **Note**
  >
  > **A free trial version of the Sumo Logic account has limited functionalities and is only available for 90 days.**

- A Cloud Syslog Source configured with your Sumo Logic account.

  For details, follow the configuration instructions under the Configure a Cloud Syslog Source section on the official Sumo Logic website.

  > **Note**
  >
  > **Transport-level security (TLS) 1.2 over TCP is required.**

- A Cloud Syslog Source Token (from the Cloud Syslog Source side).
- TLS set up on your Sumo Logic account.

  For detailed information about setting up TLS in your Sumo Logic account, see the description for setting up TLS on the Sumo Logic official website .

  > **Note**
  >
  > **After you download the `DigiCert` certificate, make sure you follow the certificate setup steps under the syslog-ng section .**

- Your Sumo Logic syslog client, configured to send data to the Sumo Logic cloud syslog service, by using AxoSyslog.

  For detailed information, follow the instructions under the Send data to cloud syslog source with syslog-ng section on the official Sumo Logic website.

- A verified connection and client configuration with the Sumo Logic service.

  > **Warning**
  >
  > **To avoid potential data loss, we strongly recommend that you verify your connection and client configuration with the Sumo Logic service before you start using the `sumologic-http()` or `sumologic-syslog()` destination with AxoSyslog in a production environment.**

- (Optional) For using the `sumologic-http()` destination, you need a HTTP Hosted Collector configured in the Sumo Logic service.

  To configure a Hosted Collector, follow the configuration instructions under the Configure a Hosted Collector section on the official Sumo Logic website.

- (Optional) For using the `sumologic-http()` destination, you need the unique HTTP collector code you receive while configuring your Host Collector for HTTP requests.

Limitations

Currently, using the `sumologic-syslog()` and `sumologic-http()` destinations with AxoSyslog has the following limitations:

- The minimum required version of AxoSyslog is version 3.27.1.
- Message format must be in RFC 5424-compliant form . Messages over 64KB in length are truncated.

  For more information about the message format limitations, see the Message format section on the official Sumo Logic website.
- 64 characters long Sumo Logic tokens must be passed in the message body.

> **Note**
>
> **Although RFC 5424 limits the structured data field (SD-ID) to 32 characters, Sumo Logic tokens are 64 characters long. If your logging client enforces the 32 characters length limit, you must pass the token in the message body.**

Declaration for the sumologic-http() destination

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

destination d_sumo_http {
    sumologic-http(
    collector("ZaVnC4dhaV3_[...]UF2D8DRSnHiGKoq9Onvz-XT7RJG2FA6RuyE5z4A==")
    deployment("eu")
    );
};
```

Declaration for the sumologic-syslog() destination

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

destination d_sumo_syslog {
    sumologic-syslog(
    token("rqf/bdxYVaBLFMoU39[...]CCC5jwETm@41123")
    deployment("eu")
    tls(peer-verify(yes) ca-dir('/etc/syslog-ng/ca.d'))
    );
};
```

8.46.1 sumologic-http()

The `sumologic-http()` and `sumologic-syslog()` destinations send log messages to Sumo Logic , a cloud-based log management and security analytics service.

Using the `sumologic-http()` destination, you can send data to the Sumo Logic service by utilizing a Hosted Collector hosted by Sumo Logic .

For the list of options, see sumologic-http() destination options.

Sending data using the sumologic-http() destination

Example: Using the sumologic-http() destination

The following example sends every log from the `system()` source to your Sumo Logic account.

```
log {
   source { system(); };

   destination {
     sumologic-http(
       collector("UNIQUE-HTTP-COLLECTOR-CODE-AS-PROVIDED-BY-sumologic")
       deployment("ENDPOINT")
       tls(peer-verify(yes) ca-dir('/etc/syslog-ng/ca.d'))
     );
   };
 };
```

8.46.2 sumologic-syslog()

The `sumologic-http()` and `sumologic-syslog()` destinations send log messages to Sumo Logic , a cloud-based log management and security analytics service.

Using the `sumologic-syslog()` destination, you can send data (both in JSON and in non-JSON format) to the Sumo Logic service.

For the list of options, see sumologic-syslog() destination options.

Sending data using the sumologic-syslog() destination

Example: Sending data using the sumologic-syslog() destination

The following example illustrates how you can use the `sumologic-syslog()` destination to send data to your Sumo Logic account.

```
log {
   source { system(); };

   destination{
     sumologic-syslog(token("USER-TOKEN-AS-PROVIDED-BY-sumologic")
       deployment("ENDPOINT")
       tls(peer-verify(required-trusted) ca-dir('/etc/syslog-ng/ca.d'))
     );
 };
 };
```

Sending JSON data using the sumologic-syslog destination

Example: Sending data using the sumologic-syslog() destination

The following example illustrates how you can use the `sumologic-syslog()` destination to send JSON data to your Sumo Logic account.

```
log {
   source{ system(); };

   destination{
     sumologic-syslog(token("USER-TOKEN-AS-PROVIDED-BY-sumologic")
       deployment("ENDPOINT")
       tls(peer-verify(required-trusted) ca-dir('/etc/syslog-ng/ca.d'))
       template("$(format-json --scope all-nv-pairs)")
     );
   };
 };
```

8.46.3 sumologic-http() and sumologic-syslog() destination options

The `sumologic-http()` and `sumologic-syslog()` destinations have the following options.

8.46.3.1 sumologic-http() destination options

The `sumologic-http()` destination supports all HTTP destination options.

In addition, the `sumologic-http()` destination also has the following options.

batch-bytes()

Accepted values:	number [bytes]
Default:	none

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()` .

Available in AxoSyslog version 3.19 and later.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number

Default:	1

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()`, set it to `0`.

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()`.

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

ca-dir()

Accepted values:	Directory name

Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

collector()

Type:	string
Default:	empty

Description: The Cloud Syslog Cloud Token that you received from the Sumo Logic service while configuring your cloud syslog source .

For details on the option in the destination's declaration, see Declaration for the sumologic-http() destination .

deployment()

Type:	string
Default:	empty string

Description: Required. This option specifies your Sumo Logic deployment .

For details on the `deployment()` option in the `sumologic-http()` destination's declaration, see Declaration for the sumologic-http() destination .

For details on the `deployment()` option in the `sumologic-syslog()` destination's declaration, see Declaration for the sumologic-syslog() destination .

headers()

Type:	string list
Default:	see description

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>`

Starting with AxoSyslog 4.18, you can use templates in the headers. If you enable batching in the destination (`batch-lines()` or `batch-bytes()`) and use message-dependent templates in `headers()` , you must set the `worker-partition-key()` option, otherwise the destination fails to start. AxoSyslog flushes the batch whenever the partition key changes, so every message of a batch has the same header values. (In versions before 4.29, the header values were calculated from the first message of the batch.)

If you want to use literal dollar signs (`$`) in `headers()` , escape them like `$$` .

> **Note**
>
> **The `headers()` option is a required option for the `sumologic-http()` destination.**

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.

In version 4.0 and newer, using the `tls()` option is optional, and Sumo Logic servers are verified using the system certificate store by default. In earlier versions, this was a required option.

8.46.3.2 sumologic-syslog() destination options

The `sumologic-syslog()` destination supports all network() destination options.

In addition, the `sumologic-syslog()` destination also has the following options.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

deployment()

Type:	string
Default:	empty string

Description: Required. This option specifies your Sumo Logic deployment .

For details on the `deployment()` option in the `sumologic-http()` destination's declaration, see Declaration for the sumologic-http() destination .

For details on the `deployment()` option in the `sumologic-syslog()` destination's declaration, see Declaration for the sumologic-syslog() destination .

port()

Type:	number
Default:	6514

Description: Optional. This option sets the port number of the Sumo Logic server to connect to.

tag()

Type:	string list
Default:	"tag"

Description: Optional. This option specifies the list of tags to add as the tags fields of Sumo Logic messages. If not specified, AxoSyslog automatically adds the tags already assigned to the message. If you set the `tag()` option, only the tags you specify will be added to the 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.

In version 4.0 and newer, using the `tls()` option is optional, and Sumo Logic servers are verified using the system certificate store by default. In earlier versions, this was a required option.

token()

Type:	string
Default:	

Description: Required option. The Cloud Syslog Cloud Token that you received from the Sumo Logic service while configuring your cloud syslog source .

8.47 syslog: Send messages to a remote logserver using the IETF-syslog protocol

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

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

Declaration:

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

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

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

> **Note**
>
> **The default ports for the different transport protocols are as follows: UDP – 514, TCP – 514, TLS – 6514.**

Example: Using the syslog() driver

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

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

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

Send the log messages using TLS encryption and use mutual authentication. For details on the encryption and authentication options, see TLS options.

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

> **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.**

8.47.1 syslog() destination options

The `syslog()` driver sends messages to a remote host (for example, an AxoSyslog 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:

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

close-on-input()

Type:	`yes` or `no`
Default:	yes

Description: By default, AxoSyslog closes destination sockets if it receives any input from the socket (for example, a reply). If this option is set to no, AxoSyslog just ignores the input, but does not close the socket.

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

failover()

Description: Available only in AxoSyslog 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, AxoSyslog waits for the a server before switching to the next failover server is set in the `time-reopen()` option.

If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

> **Warning**
>
> **The failover servers must be accessible on the same port as the primary server.**

failback()

Description: Available only in AxoSyslog version 3.17 and later.

When AxoSyslog 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, AxoSyslog behaves as follows:

- `round-robin mode` : If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

  In the following example AxoSyslog 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, AxoSyslog attempts to return to the primary server.

  After AxoSyslog 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, AxoSyslog tries to connect the primary server using the last probe.

  > **Note**
  >
  > **AxoSyslog 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.**

  In the following example AxoSyslog 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

Example: Configuring failover servers

In the following example AxoSyslog handles the logservers in round-robin fashion if the primary logserver becomes uneccassible (therefore `failback()` option is not set).

```
   destination demo_failover_roundrobin{
          syslog("primary-logserver.example.com"
                failover(
                      servers("first-failover-logserver.example.com", "second-
failover-logserver.example.com")
                )
          transport("tcp")
          port(514)
          time-reopen(60)
          );
    };
```

In the following example AxoSyslog 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 demo_failover_returntoprimary{
          syslog("primary-logserver.example.com"
                failover(
                      servers("first-failover-logserver.example.com", "second-
failover-logserver.example.com")
                      failback(
                            tcp-probe-interval(120)
                            successful-probes-required(3)
                      )
                )
          transport("tcp")
          port(514)
          time-reopen(60)
          );
    };
```

> **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 (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

UDP and other datagram transports write one message per packet.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 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 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 AxoSyslog 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 AxoSyslog.

> ### 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 AxoSyslog 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 AxoSyslog do not comply with the latest RFC which was published after the release of AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 using AxoSyslog 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.

The maximum size of spoofed datagrams in udp() destinations is set to 1024 bytes by default. To change the maximum size, use the `spoof-source-max-msglen()` option.

> **Note**
>
> Anything above the size of the maximum transmission unit (MTU), which is 1500 bytes by default, is not recommended because of fragmentation.
>
> The maximum datagram in IP protocols (both IPv4 and IPv6) is 65535 bytes including the IP and UDP headers. The minimum size of the IPv4 header is 20 bytes, the IPv6 is 40 bytes, and the UDP is 8 bytes.

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), AxoSyslog 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 AxoSyslog waits for identical messages.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

> **Warning**
>
> 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 AxoSyslog 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.

Warning

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 AxoSyslog 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.

Warning

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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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, AxoSyslog 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.**

8.48 syslog-ng(): Forward logs to another syslog-ng node

The `syslog-ng()` destination driver forwards log messages to another AxoSyslog node in EWMM format.

> Note: For similar functionality using the OpenTelemetry protocol, see axosyslog-otlp(): Forward logs to another node using OpenTelemetry.

The Enterprise-wide message model or EWMM allows you to deliver structured messages from the initial receiving AxoSyslog 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 AxoSyslog component in your infrastructure has received it, which is important if you want to forward a message for example, to a SIEM system. To make use of the enterprise-wide message model, you have to use the `syslog-ng()` destination on the sender side , and the `default-network-drivers()` source on the receiver side .

The following is a sample log message in EWMM format.

```
<34>1 2026-10-11T22:14:15+00:00 my-host @syslog-ng - - -
{"PROGRAM":"su","MESSAGE":"'su root' failed for lonvick on
/dev/pts/8","HOST":"mymachine","HOST_FROM":"my-host","._TAGS":
[".source.s_network"]}
```

Prerequisites

- AxoSyslog version 3.16 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

- The node that receives this message must use the default-network-drivers() source to properly handle the messages.

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.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

Warning

When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.

AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

Note

If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

Type:	number(messages)
Default:	1000

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

Warning

Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

failover()

Description: Available only in AxoSyslog 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, AxoSyslog waits for the a server before switching to the next failover server is set in the `time-reopen()` option.

If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

> **Warning**
>
> **The failover servers must be accessible on the same port as the primary server.**

failback()

Description: Available only in AxoSyslog version 3.17 and later.

When AxoSyslog 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, AxoSyslog behaves as follows:

- `round-robin mode` : If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

  In the following example AxoSyslog 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, AxoSyslog attempts to return to the primary server.

  After AxoSyslog 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, AxoSyslog tries to connect the primary server using the last probe.

  > **Note**
  >
  > **AxoSyslog 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.**

  In the following example AxoSyslog 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

Example: Configuring failover servers

In the following example AxoSyslog handles the logservers in round-robin fashion if the primary logserver becomes uneccassible (therefore `failback()` option is not set).

```
   destination demo_failover_roundrobin{
          syslog("primary-logserver.example.com"
                failover(
                      servers("first-failover-logserver.example.com", "second-
failover-logserver.example.com")
                )
          transport("tcp")
          port(514)
          time-reopen(60)
          );
    };
```

In the following example AxoSyslog 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 demo_failover_returntoprimary{
          syslog("primary-logserver.example.com"
                failover(
                      servers("first-failover-logserver.example.com", "second-
failover-logserver.example.com")
                      failback(
                            tcp-probe-interval(120)
                            successful-probes-required(3)
                      )
                )
          transport("tcp")
          port(514)
          time-reopen(60)
          );
    };
```

> **Note**
>
> **This option is not available for the connection-less UDP protocol, because in this case the client does not detect that the destination becomes inaccessible.**

Example: Specifying failover servers for syslog() destinations

The following example specifies two failover servers for a simple syslog() destination.

```
   destination d_syslog_tcp{
        syslog("10.100.20.40"
            transport("tcp")
            port(6514)
            failover-servers("10.2.3.4", "myfailoverserver")
        );
    };
```

The following example specifies a failover server for a network() destination that uses TLS encryption.

```
destination d_syslog_tls{
     network("10.100.20.40"
         transport("tls")
         port(6514)
         failover-servers("10.2.3.4", "myfailoverserver")
         tls(peer-verify(required-trusted)
         ca-dir('/opt/syslog-ng/etc/syslog-ng/keys/ca.d/')
         key-file('/opt/syslog-ng/etc/syslog-ng/keys/client_key.pem')
         cert-file('/opt/syslog-ng/etc/syslog-ng/keys/client_certificate.pem'))
     );
 };
```

flags()

Type:	no-multi-line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The `no-multi-line` flag disables line-breaking in the messages: the entire message is converted to a single line.
- *syslog-protocol*: The `syslog-protocol` flag instructs the driver to format the messages according to the new IETF syslog protocol standard (RFC5424), but without the frame header. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the `syslog` driver, and that the `syslog` driver automatically adds the frame header to the messages.

flush-lines()

Type:	number
Default:	Use global setting (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

ip-protocol()

Type:	number

Default:	4

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

Note that listening on a port using IPv6 automatically means that you are also listening on that port using IPv4. That is, if you want to have receive messages on an IP-address/port pair using both IPv4 and IPv6, create a source that uses the `ip-protocol(6)` . You cannot have two sources with the same IP-address/port pair, but with different `ip-protocol()` settings (it causes an `Address already in use` error).

For example, the following source receives messages on TCP, using the `network()` driver, on every available interface of the host on both IPv4 and IPv6.

```
source s_network_tcp { network( transport("tcp") ip("::") ip-protocol(6) port(601)
); };
```

ip-tos()

Type:	number
Default:	0

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

ip-ttl()

Type:	number
Default:	0

Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type:	yes or no
Default:	yes

Description: Specifies whether connections to destinations should be closed when `syslog-ng` is reloaded. Note that this applies to the client (destination) side of the 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 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 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 AxoSyslog do not comply with the latest RFC which was published after the release of AxoSyslog 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 AxoSyslog server.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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), AxoSyslog 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 AxoSyslog waits for identical messages.

tcp-keepalive-intvl()

Type:	number [seconds]
Default:	0

Description: Specifies the interval (number of seconds) between subsequent 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.

> **Warning**
>
> 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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog 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.

> **Warning**
>
> 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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

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, AxoSyslog 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.**

8.49 axosyslog-otlp(): Forward logs to another node using OpenTelemetry

The `axosyslog-otlp()` source and destination allows you to transfer the internal representation of log messages between AxoSyslog instances using the OpenTelemetry protocol. Unlike the traditional AxoSyslog drivers that rely on simple TCP connections, `axosyslog-otlp()` leverages the OpenTelemetry protocol for efficient and reliable log message transmission.

The key benefits of the `axosyslog-otlp()` drivers are:

- scalability (via the `workers()` option),
- built-in application layer acknowledgement,
- support for Google service authentication (ADC or ALTS), and
- improved load balancing capabilities.

To use it, configure an `axosyslog-otlp()` destination on the sender node, and an `axosyslog-otlp()` source on the receiver node, like this:

```
destination d_axosyslog_otlp {
  axosyslog-otlp(url("your-receiver-axosyslog-instance:4317"));
};
```

```
source s_axosyslog_otlp {
  axosyslog-otlp(port(4317));
};
```

Prerequisites

- AxoSyslog version 4.12 or later. (From version 4.4 to 4.11, this driver was called `syslog-ng-otlp()` .)
- This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Options

The `axosyslog-otlp()` destination has the following options.

auth()

You can set authentication in the `auth()` option of the driver. By default, authentication is disabled (`auth(insecure())`).

The following authentication methods are available in the `auth()` block:

adc()

Application Default Credentials (ADC) . This authentication method is only available for destinations.

service-account-key()

Available in AxoSyslog version 4.15 and later.

Use the specified service account key for ADC authentication. File path must be the absolute path. For example:

```
auth(adc(service-account-key("absolute-path-to-key-file")))
```

alts()

Application Layer Transport Security (ALTS) is a simple to use authentication, only available within Google's infrastructure. It accepts the `target-service-account()` option, where you can list service accounts to match against when authenticating the server.

```
  axosyslog-otlp(
    port(4317)
    auth(alts())
  );
```

insecure()

This is the default method, authentication is disabled (`auth(insecure())`).

tls()

> **Note**
>
> **gRPC-based drivers have a different `tls()` block implementation from the `network()` or `http()` drivers, but most features are the same.**

```
destination d_otlp {
    axosyslog-otlp(
      url("your-otel-server:12346")
      auth(
        tls(
          ca-file("/path/to/ca.pem")
          key-file("/path/to/key.pem")
          cert-file("/path/to/cert.pem")
        )
      )
    );
  };
```

`tls()` accepts the following options.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

batch-bytes()

Accepted values:	number [bytes]
Default:	4MB

Available in AxoSyslog version 4.6 and later.

Description: Sets the maximum size of payload in a batch. If the size of the messages reaches this value, AxoSyslog sends the batch to the destination even if the number of messages is less than the value of the `batch-lines()` option. The batch might be at most 1 message larger than the set limit.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-bytes()`.

OTLP has a default 4 MiB batch limit, therefore the default value for `batch-bytes()` is 4 MB, which is a bit below 4 MiB.

The batch size is calculated before compression, which is the same as the limit is calculated on the server.

batch-idle-timeout()

Type:	time in milliseconds
Default:	`0` (disabled)

Available in AxoSyslog 4.24 and later.

Description: `batch-idle-timeout()` measures the elapsed time since the last message was added to the batch (`batch-timeout()` defines the maximum time used to collect a batch, starting from the first message). If either `batch-idle-timeout()` or `batch-timeout()` expires, AxoSyslog closes and sends the batch.

batch-lines()

Type:	number
Default:	0

Description: Specifies how many lines are flushed to a destination in one batch. The AxoSyslog 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. To disable `batch-lines()` , set it to `0` .

For example, if you set `batch-lines()` to 100, AxoSyslog waits for 100 messages.

If the `batch-timeout()` option is disabled, the AxoSyslog application flushes the messages if it has sent `batch-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

Note that if the `batch-timeout()` option is enabled and the queue becomes empty, AxoSyslog flushes the messages only if `batch-timeout()` expires, or the batch reaches the limit set in `batch-lines()` .

For optimal performance, make sure that the AxoSyslog source that feeds messages to this destination is configured properly: the value of the `log-iw-size() / max-connections()` of the source must be higher than the `batch-lines() * workers()` of the destination. Otherwise, the size of the batches cannot reach the `batch-lines()` limit.

batch-timeout()

Type:	time in milliseconds
Default:	`-1` (disabled)

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends batches to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `batch-timeout()` milliseconds.

channel-args()

Type:	arrow list
Default:	-

Description: The `channel-args()` option is available in gRPC-based drivers. It accepts name-value pairs and sets channel arguments defined in the GRPC Core library documentation . For example:

```
channel-args(
    "grpc.loadreporting" => 1
    "grpc.minimal_stack" => 0
)
```

compression()

Type:	boolean

Default:	`no`

Available in AxoSyslog version 4.5.0 and later.

Description: Enables compression in gRPC requests. Although gRPC supports various compression methods, currently only deflate is supported (which is basically the same as gzip).

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:

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**

Default: no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type: string

Default: N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	yes/no
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

Warning

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.

truncate-size-ratio()

Type:	**number((between 0 and 1))**
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

Warning

Axoflow does not recommend you to change `truncate-size-ratio()` . Only change its value if you understand the performance implications of doing so.

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(2000000)
            reliable(no)
            dir("/tmp/disk-buffer")
        )
    );
};
```

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the

original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `drop-property`: Omit the affected property (macro, template, or message-field) from the log message and log an error message to the `internal()` source.
- `fallback-to-string`: Convert the property to string and log an error message to the `internal()` source.
- `silently-drop-message`: Drop the entire message silently, without logging the error.
- `silently-drop-property`: Omit the affected property (macro, template, or message-field) silently, without logging the error.
- `silently-fallback-to-string`: Convert the property to string silently, without logging the error.

persist-name()

Type:	string
Default:	N/A

Description: If you receive the following error message during AxoSyslog 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: If AxoSyslog cannot send a message, it will try again until the number of attempts reaches `retries()`.

If the number of attempts reaches `retries()`, AxoSyslog will wait for `time-reopen()` time, then tries sending the message again.

send-time-zone()

Accepted values:	name of the timezone, or the timezone offset
Default:	local timezone

Description: Specifies the time zone associated with the messages sent by `syslog-ng`, if not specified otherwise in the message or in the destination driver. For details, see Timezones and daylight saving.

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

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

url()

Type:	string
Default:	`localhost:9095`

Description: The URL of the AxoSyslog receiver.

worker-partition-autoscaling()

Type:	`yes` , `no`
Default:	`no`

Available in version 4.21 and later.

When using `worker-partition-key()` to categorize messages into different batches, the messages are hashed into workers by default. This prevents distributing across workers based on load.

Setting `worker-partition-autoscaling(yes)` uses a 1-minute statistic to distribute high-traffic partitions among multiple workers, allowing each worker to maximize its batch size. When using `worker-partition-autoscaling(yes)` , set the number of `workers()` to higher than the expected number of partitions.

worker-partition-buckets()

Type:	template
Default:	

Description: The `worker-partition-buckets()` option determines the number of worker threads used for the `worker-partition-key()`. Note that the number set by `worker-partition-buckets()` should be lower than the number of `workers()`.

worker-partition-key()

Type:	template
Default:	

Description: The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. When batching is enabled (`batch-lines()` or `batch-bytes()`) and the `url()`, `headers()`, or `body-prefix()` options contain message-dependent templates, `worker-partition-key()` is required: without it, the destination fails to start with an error. Set it to a template that contains all the message-dependent templates used in these options. AxoSyslog flushes the batch whenever the partition key changes, so every batch contains messages with identical URL, headers, and body prefix.

Templates that resolve to the same value for every message (for example, ones that contain only literals or `$(url-encode literal)`) don't require `worker-partition-key()`.

For example, you can partition messages based on the destination host:

```
worker-partition-key("$HOST")
```

workers()

Type:	integer
Default:	1

Description: Specifies the number of worker threads (at least 1) that AxoSyslog uses to send messages to the server. Increasing the number of worker threads can drastically improve the performance of the destination.

Warning

`Hazard of data loss.` **When you use more than one worker threads together with disk-based buffering, AxoSyslog creates a separate disk buffer for each worker thread. This means that decreasing the number of workers can result in losing data currently stored in the disk buffer files. Do not decrease the number of workers when the disk buffer files are in use.**

8.50 tcp, tcp6, udp, udp6: OBSOLETE - Send 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: Collect messages using the RFC3164 protocol (network() driver) and network: Send 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, an AxoSyslog 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.

8.50.1 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: Collect messages using the RFC3164 protocol (network() driver) and network: Send 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.

8.50.1.1 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')
        )
    );
};
```

8.51 telegram: Send messages to Telegram

The `telegram()` destination sends log messages to Telegram , which is a secure, cloud-based mobile and desktop messaging app.

Note that this destination automatically uses the certificate store of the system (for details, see the curl documentation).

Prerequisites

- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

- This feature requires a separate module. Install the `axosyslog-mod-http` package on Debian/Ubuntu , or the `axosyslog-http` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

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

telegram(parameters);
```

You can use the `proxy()` option to configure the HTTP driver in all HTTP-based destinations to use a specific HTTP proxy that is independent from the proxy configured for the system.

Example: Using the telegram() driver

The following example creates a `telegram()` destination.

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

destination d_telegram {
    telegram(
        template("${MESSAGE}")
        throttle(1)
        parse-mode("markdown")
        disable-web-page-preview("true")
        bot-id("<bot id>")
        chat-id("<chat id>")
    );
};
```

8.51.1 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_notification()

Type:	boolean
Default:	`false`

Description: Enables the `telegram()` destination to send silent messages. By default, the `disable_notification()` value is `false`.

Example: using the disable_notification() option with the telegram() destination

The following example illustrates how you can configure the `disable_notification()` option to send silent messages to the `telegram()` destination.

```
destination {
   telegram(
     bot-id(...)
     chat-id(...)
     disable_notification(true)
   );
 };
```

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 a security point of view, Axoflow 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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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:	`${MESSAGE}`

Description: Specifies the content of the message. The AxoSyslog application will automatically encode the content of this option using the `url-encode()` template function.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.

8.52 unix-stream, unix-dgram: Send 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"); };
```

8.52.1 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:

close-on-input()

Type:	`yes` or `no`
Default:	yes

Description: By default, AxoSyslog closes destination sockets if it receives any input from the socket (for example, a reply). If this option is set to no, AxoSyslog just ignores the input, but does not close the socket.

create-dirs()

Type:	yes or no
Default:	no

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

disk-buffer()

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

capacity-bytes()

Type:	**number (bytes)**
Default:	1MiB

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.

Starting with AxoSyslog version 4.29, you can change the `capacity-bytes()` of an existing disk-buffer file. The new value takes effect when you reload or restart AxoSyslog, if the content of the disk-buffer allows it:

- The new value takes effect on the next reload or restart, or later, once the content of the disk-buffer allows it.
- When you decrease the size, the change is applied only if the messages in the file fit into the new size. In this case, AxoSyslog truncates the file to the new size.

Otherwise, AxoSyslog logs a warning, and applies the new value later, when the content of the disk-buffer allows it. When you increase the size and `prealloc(yes)` is set, AxoSyslog preallocates the file to the new size.

In AxoSyslog version 4.2 and earlier, this option was called `disk-buf-size()`.

compaction()

Type:	**yes/no**
Default:	no

Description: If set to `yes`, AxoSyslog prunes the unused space in the LogMessage representation, making the disk queue size smaller at the cost of some CPU time. Setting the `compaction()` argument to `yes` is recommended when numerous name-value pairs are unset during processing, or when the same names are set multiple times.

> **Note**
>
> **Simply unsetting these name-value pairs by using the `unset()` rewrite operation is not enough, as due to performance reasons that help when AxoSyslog is CPU bound, the internal representation of a `LogMessage` will not release the memory associated with these name-value pairs. In some cases, however, the size of this overhead becomes significant (the raw message size can grow up to four times its original size), which unnecessarily increases the disk queue file size. For these cases, the compaction will drop `unset` values, making the `LogMessage` representation smaller at the cost of some CPU time required to perform compaction.**

dir()

Type:	**string**
Default:	N/A

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

> **Warning**
>
> When creating a new `dir()` option for a disk buffer, or modifying an existing one, make sure you delete the persist file.
>
> AxoSyslog 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, AxoSyslog will look for or create disk-buffer files in their old location. To ensure that AxoSyslog 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.

> **Note**
>
> **If the `dir()` path provided by the user does not exist, AxoSyslog creates the path with the same permission as the running instance.**

flow-control-window-bytes()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-size()`.

flow-control-window-size()

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`.

In AxoSyslog version 4.2 and earlier, this option was called `mem-buf-length()`.

front-cache-size()

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

Description: The number of messages stored in the output buffer of the destination. Note that if you change the value of this option and the disk-buffer already exists, the change will take effect when the disk-buffer becomes empty.

Options `reliable()` and `capacity-bytes()` are required options.

In AxoSyslog version 4.2 and earlier, this option was called `qout-size()`.

prealloc()

Type:	**yes/no**
Default:	no

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

Available in AxoSyslog 4.0 and later.

reliable()

Type:	yes/no
Default:	no

Description: If set to `yes`, AxoSyslog cannot lose logs in case of reload/restart, unreachable destination or AxoSyslog 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.

> **Warning**
>
> **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.**

truncate-size-ratio()

Type:	number((between 0 and 1))
Default:	1 (do not truncate)

Description: Limits the truncation of the disk-buffer file. Truncating the disk-buffer file can slow down the disk IO operations, but it saves disk space. By default, AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default (`truncate-size-ratio(1)`). Earlier versions freed the disk-space when at least 10% of the disk-buffer file could be freed (`truncate-size-ratio(0.1)`).

AxoSyslog only truncates the file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()` .

- Smaller values free disk space quicker.
- Larger ratios result in better performance.

If you want to avoid performance fluctuations:

- use `truncate-size-ratio(1)` (never truncate), or
- use `prealloc(yes)` to reserve the entire size of the disk-buffer on disk .

> **Warning**
>
> **Axoflow does not recommend you to change `truncate-size-ratio()`. Only change its value if you understand the performance implications of doing so.**

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(
            flow-control-window-bytes(10000)
            capacity-bytes(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(
            flow-control-window-size(10000)
            capacity-bytes(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 (exception: for http() destination, the default is 1).

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

For optimal performance when sending messages to an AxoSyslog server, make sure that the value of `flush-lines()` is smaller than the window size set in the `log-iw-size()` option in the source of your server.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

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 AxoSyslog 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 AxoSyslog to execute external applications.**

Using `hook-commands()` when AxoSyslog starts or stops

To execute an external program when AxoSyslog starts or stops, use the following options:

`startup()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog starts.

`shutdown()`

Type:	**string**
Default:	N/A

Description: Defines the external program that is executed as AxoSyslog stops.

Using the hook-commands() when AxoSyslog reloads

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

`setup()`

Type:	**string**
Default:	N/A

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

`teardown()`

Type:	**string**
Default:	N/A

Description: Defines an external program that is executed when the AxoSyslog configuration is stopped or torn down, for example, on shutdown or during a AxoSyslog reload.

Example: Using hook-commands() with a network source

In the following example, the `hook-commands()` is used with the `network()` driver and it opens an iptables port automatically as AxoSyslog 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 AxoSyslog 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 connections, server-side (source) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the source.

mark-freq()

Accepted values:	number [seconds]
Default:	1200

Description: An alias for the obsolete `mark()` option, retained for compatibility with 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 3.4 and later.

so-broadcast()

Type:	yes or no
Default:	no

Description: This option controls the `SO_BROADCAST` socket option required to make AxoSyslog 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), AxoSyslog 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 AxoSyslog 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 AxoSyslog. 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 backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

throttle()

Type:	number
Default:	0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying `0` or a lower value sets the output limit to

unlimited.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

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.**

8.53 usertty: Send messages to a user terminal

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);
```

Example: Using the usertty() driver

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

escaping()

Accepted values:	`yes` or `no`
Default:	`no`

Available in AxoSyslog 4.26 and later.

Description: Turns on escaping for the `'`, `"`, and backslash characters from the messages. This behavior is useful when the messages are passed to an application that cannot handle escaped characters properly.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

8.54 Write your own custom destination in Java or Python

The AxoSyslog 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 AxoSyslog in C (the main language of AxoSyslog), or using its language bindings, for example, Java or Python. .

- For details on extending AxoSyslog in Python, see the python: Write custom Python destinations.
- For details on extending AxoSyslog in Java, see the AxoSyslog 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.**

8.55 Client-side failover

AxoSyslog 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, AxoSyslog switches to the third server in the list, and so on. If there are no more failover servers left, AxoSyslog 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. AxoSyslog nominates this destination over the failover servers, and handles it as the primary address.

When AxoSyslog 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, AxoSyslog behaves as follows:

- `round-robin mode`: If `failback()` is not set, AxoSyslog does not attempt to return to the primary server even if it becomes available. In case the failover server fails, AxoSyslog attempts to connect the next failover server in the list in round-robin fashion.

  In the following example AxoSyslog 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, AxoSyslog attempts to return to the primary server.

    After AxoSyslog 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, AxoSyslog tries to connect the primary server using the last probe.

> **Note**
>
> **AxoSyslog 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.**

In the following example AxoSyslog 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 AxoSyslog is restarted, it attempts to connect the primary server.

If AxoSyslog uses TLS-encryption to communicate with the remote server, AxoSyslog 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 AxoSyslog 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.

9 log: Filter and route log messages using log paths, flags, and filters

Build log paths that route messages from sources to destinations, select messages with filters, and manage flow-control and disk buffers.

FilterX or classic filters, parsers, and rewrite rules?

FilterX replaces filters, parsers, and rewrite rules with a single typed language that can also handle nested JSON and other complex data. You don't have to migrate: the classic blocks keep working, and you can use both in the same log path. To convert your existing configuration, see Update to FilterX.

9.1 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...]);
 };
```

Warning

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

Note

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

Named log paths and log path metrics

In AxoSyslog version 4.1 and later, you can add an ID or name to the log path to make the configuration file more readable. Also, AxoSyslog collects ingress and egress metrics for named log paths. For example:

```
log top-level {
    source(s_local);

    log inner-1 {
        filter(f_inner_1);
        destination(d_local_1);
    };

    log inner-2 {
        filter(f_inner_2);
        destination(d_local_2);
    };
};
```

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);
    };
```

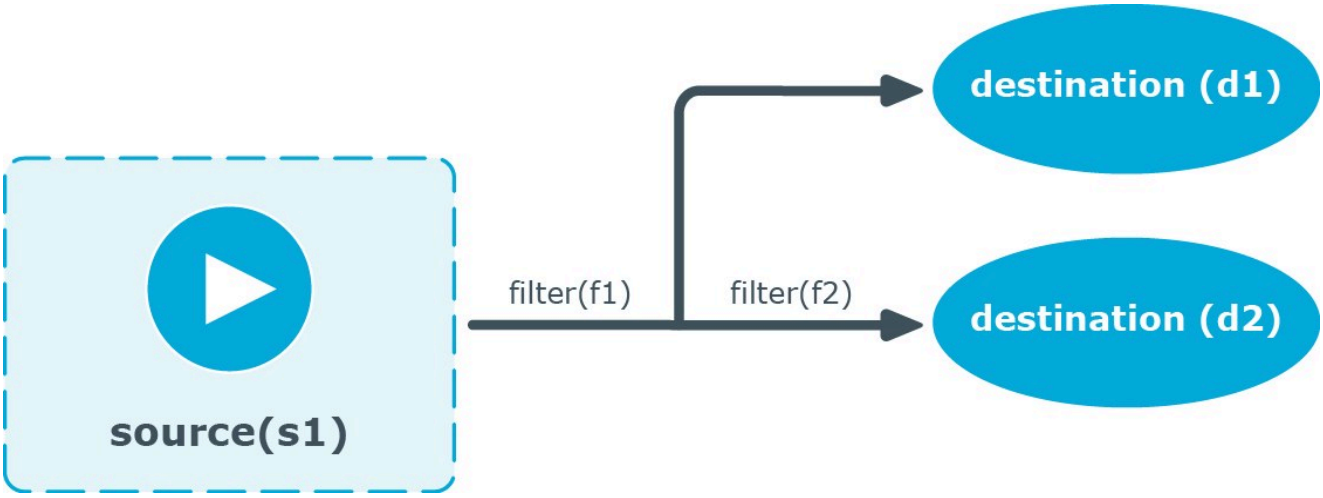
All matching log statements are processed by default, and the messages are sent to *every* matching destination by default. So a single log message might be sent to the same destination several times, provided the destination is listed in several log statements, and it can be also sent to several different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

9.1.1 Embedded log statements

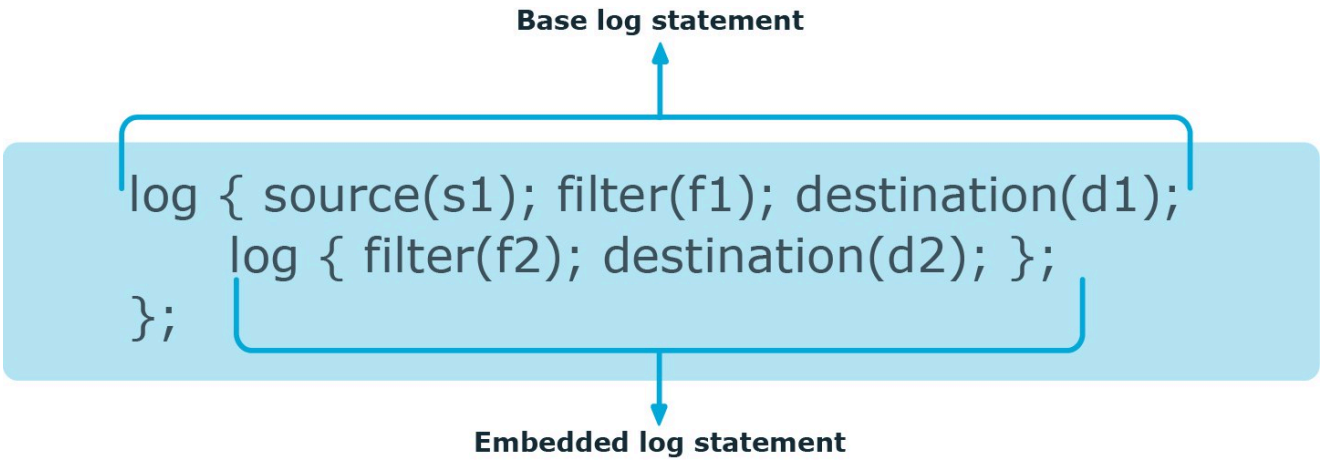
Starting from version 3.0, AxoSyslog can handle embedded log statements (also called log pipes). Embedded log statements are useful for creating complex, multi-level log paths with several destinations and use filters, parsers, and rewrite rules.

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.



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.



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.

9.1.1.1 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);
     };
 };
```

9.1.2 if-else-elif: Conditional expressions

You can use `if {}`, `elif {}`, and `else {}` blocks to configure conditional expressions.

Note that `if {}` blocks imply the `flags(final)` option automatically, and are equivalent to `log { some condition; flags(final);}`.

Conditional expressions' format

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.

Using the if {} and else {} blocks in your configuration

You can copy-paste the following example and use it as a template for using the `if {}` and `else {}` blocks in your configuration.

```
log{
    source { example-msg-generator(num(1) template("...,STRING-TO-MATCH,..."));};
    source { example-msg-generator(num(1) template("...,NO-MATCH,..."));};

    if (message("STRING-TO-MATCH")) {
        destination { file(/dev/stdout template("matched: $MSG\n") persist-
name("1")); };
    }

    else {
    destination { file(/dev/stdout template("unmatched: $MSG\n") persist-
name("2")); };
    };
};
```

The configuration results in the following console printout:

```
matched: ...,STRING-TO-MATCH,...
unmatched: ...,NO-MATCH,...
```

An alternative, less straightforward way to implement conditional evaluation is to use junctions. For details on junctions and channels, see Junctions and channels.

9.1.3 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 AxoSyslog, 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**
>
> **AxoSyslog has several parsers that you can use to parse non-compliant messages. You can even write a custom 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.

9.1.4 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.

Log statement flags

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, AxoSyslog 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 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 (AxoSyslog 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 AxoSyslog 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, AxoSyslog will drop messages if the destination queues are full. If enabled, AxoSyslog will only drop messages if the destination queues/window sizes are improperly sized. For details, see Managing incoming and outgoing messages with flow-control. Starting with version 4.12, you can enable flow-control for every log path using the `log-flow-control()` global option .

no-flow-control

Available in AxoSyslog 4.12 and later.

Disables flow-control for a log path if the `log-flow-control()` global option is enabled.

Using the `flow-control` and `no-flow-control` flags, you can enable/disable flow-control just the way you want. This can be especially useful for multi-level Embedded log statements.

> **Warning**
>
> **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 AxoSyslog 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 AxoSyslog 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, AxoSyslog treats the message as 'processed'. Since the first log path includes the `final` flag, AxoSyslog will not pass the message to the second log path (the one with the `fallback` flag). As a result, AxoSyslog 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 AxoSyslog 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') };
};
...
```

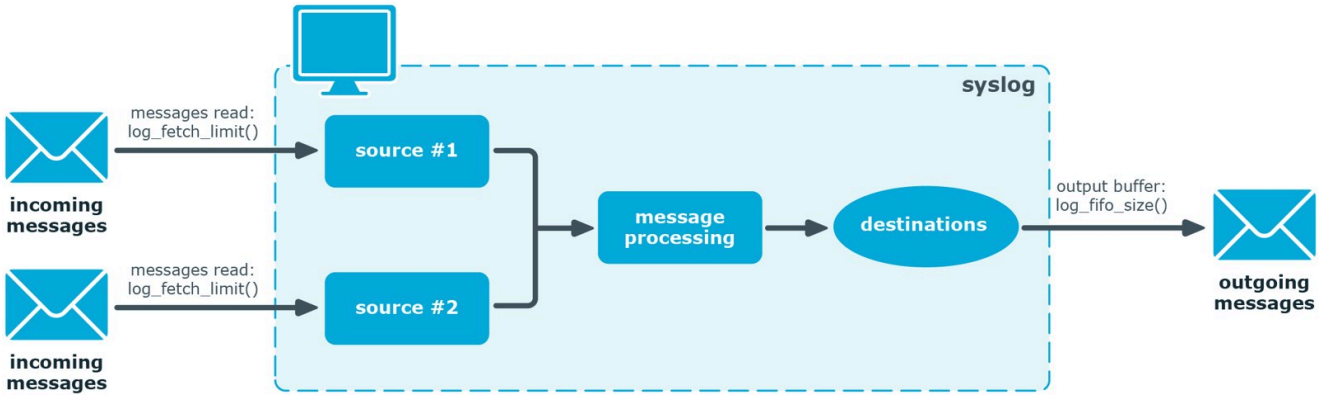
(Without the `drop-unmatched` flag, AxoSyslog would check if the message complies with the second `if` condition, that is, whether or not the message contains the string `bar`.)

9.2 Managing incoming and outgoing messages with flow-control

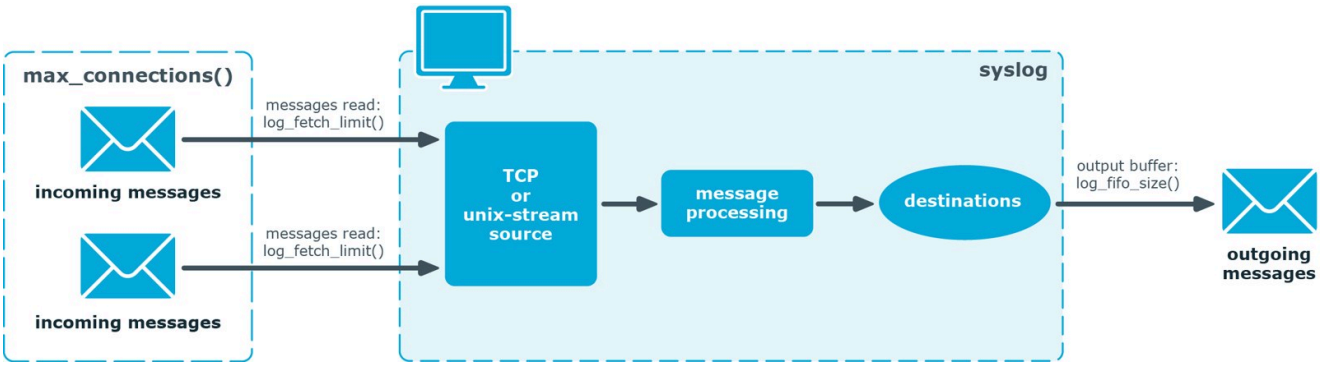
This section describes the internal message-processing model of `syslog-ng`, as well as the flow-control feature that can prevent message losses.

The AxoSyslog 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, AxoSyslog polls every source and reads the available messages. These messages are processed and put into the output buffer of AxoSyslog (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 AxoSyslog 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.



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



Log paths without flow-control

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.

Log paths with flow-control

The AxoSyslog application uses flow-control in the following cases:

- Hard flow-control: the `flow-control` flag is enabled for the particular log path, or the `log-flow-control()` global option is set. You can selectively disable flow control for individual log paths using the `no-flow-control` flag.

  Using the `flow-control` and `no-flow-control` flags, you can enable/disable flow-control just the way you want. This can be especially useful for multi-level Embedded log statements.

- Soft flow-control: the log path includes a file destination.

> **Note**
>
> **The way flow-control works has changed significantly in version AxoSyslog 3.22. If you are using an older version of AxoSyslog, consult the documentation of the version you are using for details about flow-control.**

The flow-control of AxoSyslog introduces a control window to the source that tracks how many messages can AxoSyslog accept from the source. Every message that AxoSyslog reads from the source lowers the window size by one, every message that AxoSyslog successfully sends from the output buffer increases the window size by one. If the window is full (that is, its size decreases to zero), AxoSyslog stops reading messages from the source. The initial size of the control window is by default `100`. If a source accepts messages from multiple connections, all messages use the same control window.

When using flow-control, AxoSyslog automatically sets the size of the output buffer so that it matches the size of the control window of the sources. Note that starting with AxoSyslog 3.22, `log-fifo-size()` only affects log paths that are not flow-controlled.

> **Note**
>
> If the source can handle multiple connections (for example, `network()` and `syslog()`), 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.
>
> For UDP-based sources, the window size is not divided by `max-connections()`, because UDP-based traffic is treated as a single connection for the source. Also, flow control has no effect on UDP-based traffic, because there's no way to notify the peers that their messages cannot be processed. For UDP sources, we recommend configuring appropriately sized disk buffers .

Dynamic flow-control

In addition to the static control window set using the `log-iw-size()` option, you can also allocate a dynamic window to the source. The AxoSyslog application uses this window to dynamically increase the static window of the active connections. The dynamic window is distributed evenly among the active connections of the source. The AxoSyslog application periodically checks which connections of the source are active, and redistributes the dynamic window. If only one of the connections is active, it receives the entire dynamic window, while other connections receive only their share of the static window.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.
>
> If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Using dynamic flow-control on your AxoSyslog server is useful when the source has lots of connections, but only a small subset of the active clients send messages at high rate, and the memory of the AxoSyslog server is limited. In other cases, it is currently not recommended, because it can result in higher memory usage and fluctuating performance compared to using only the static window.

When flow-control is used, every source has its own control window. As a worst-case situation, memory of the host must be greater than the total size of the messages of every control window, plus the size of the dynamic window, that is, the `log-iw-size()` + `dynamic-window-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 memory of the host. Otherwise, some messages might not fit in the memory, and messages may be lost.

If dynamic flow-control is disabled (which is the default behavior), the value of the `log-iw-size()` option cannot be lower than 100. If dynamic flow-control is enabled, you can decrease the value of the `log-iw-size()` option (to the minimum of 1).

In case of soft flow-control there is no message lost if the destination can accept messages. It is possible to lose messages if it cannot accept messages (for example, the file destination is not writable, or the disk becomes full), and all buffers are full. Soft flow-control cannot be configured, it is automatically available for file destinations.

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: 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 {
     source(s_file);
     destination(d_file);
     destination(d_tcp);
 };
```

Warning

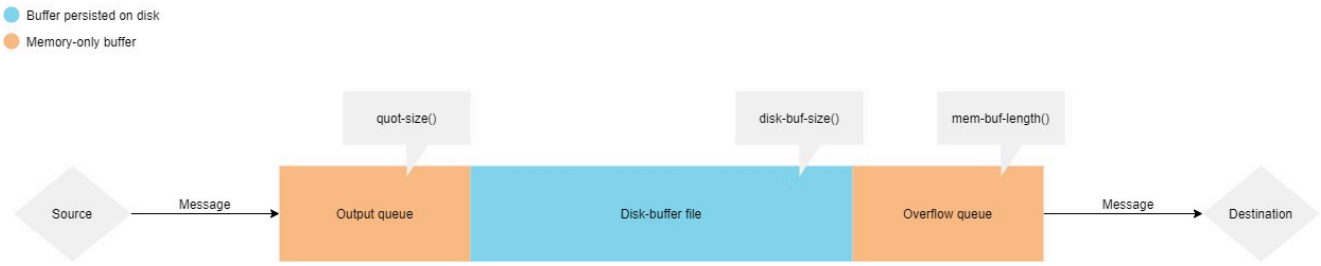
Hazard of data loss! For destinations other than file, soft flow-control is not available. Thus, it is possible to lose log messages on those destinations. To avoid data loss on those destinations, use hard flow-control.

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 {
     source(s_file);
     destination(d_file);
     destination(d_tcp);
     flags(flow-control);
 };
```

Handling outgoing messages

The AxoSyslog application handles outgoing messages the following way:



- *Output queue*: Messages from the output queue are sent to the target AxoSyslog server. The AxoSyslog 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, AxoSyslog 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, AxoSyslog 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, unless flow-control is enabled. When dynamic flow-control is enabled, AxoSyslog sets the size of the overflow queue automatically. For details on sizing the `log-fifo-size()` parameter, see Configuring flow-control.

9.2.1 Flow-control and multiple destinations

Using flow-control on a source has an important side-effect if the messages of the source are sent to multiple destinations. If flow-control is in use and one of the destinations cannot accept the messages, the other destinations do not receive any messages either, because AxoSyslog 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 (`capacity-bytes()` minus `flow-control-window-bytes()`).

9.2.2 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 AxoSyslog application normally reads a maximum of `log-fetch-limit()` number of messages from a source.
- From TCP and unix-stream sources, AxoSyslog 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. The size of this buffer is set automatically for log paths that use flow-control, and can be set using the `log-fifo-size()` option for other log paths.
- 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()` option sets the size of the static control window. Optionally, you can enable a dynamic control window for the source using the `dynamic-window-size()` option.
- 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 dynamic control window is automatically distributed among the active connections of the source.
- If the control window is full, AxoSyslog stops reading messages from the source until some messages are successfully sent to the destination.
- If the output buffer becomes full, and neither disk-buffering nor flow-control is used, messages may be lost.
- You can explicitly enable flow control:
  - using the `flags(flow-control)` option for a log path, or
  - using the `log-flow-control()` global option for every log path. In this case, you can selectively disable flow control for individual log paths using the `no-flow-control` flag.

  Using the `flow-control` and `no-flow-control` flags, you can enable/disable flow-control just the way you want. This can be especially useful for multi-level Embedded log statements.

Warning

If you modify the `max-connections()` or the `log-fetch-limit()` parameter, do not forget to adjust the `log-iw-size()` and `dynamic-window-size()` parameters accordingly.

CAUTION:

If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Example: Sizing parameters for flow-control

Suppose that AxoSyslog has a source that must accept up to 300 parallel connections. Such situation can arise when a network source receives connections from many clients, or if many applications log to the same socket.

Set the `max-connections()` parameter of the source to `300` . However, the `log-fetch-limit()` (default value: 10) parameter applies to every connection of the source individually, while the `log-iw-size()` (default value: 1000) parameter applies to the source. In a worst-case scenario, the destination does not accept any messages, while all 300 connections send at least `log-fetch-limit()` number of messages to the source during every poll loop. Therefore, the control window must accommodate at least `max-connections()` * `log-fetch-limit()` messages to be able to read every incoming message of a poll loop. In the current example this means that `log-iw-size()` should be greater than `300*10=3000` . If the control window is smaller than this value, the control window might fill up with messages from the first connections – causing AxoSyslog 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, AxoSyslog 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);
 };
```

9.3 Using disk-based and memory buffering

The AxoSyslog 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 AxoSyslog 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, AxoSyslog 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 destination driver supports the `disk-buffer()` option. The `network()`, `syslog()`, `tcp()`, and `tcp6()` destination drivers can also use the disk-buffer option, except when using the `udp` transport method.

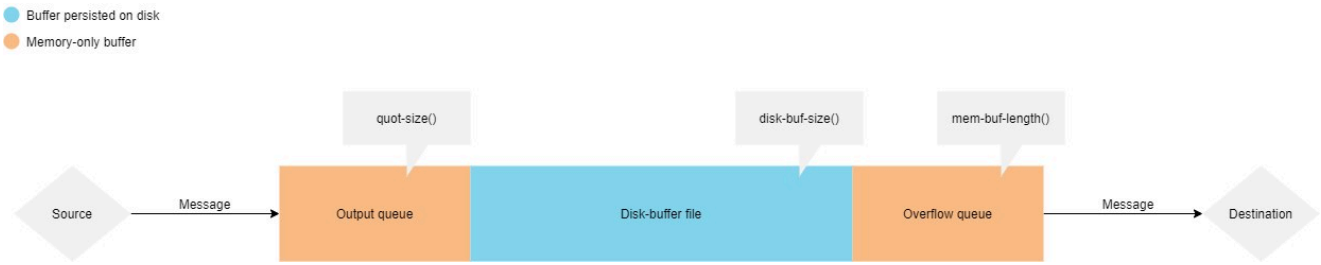
Every such destination uses a separate disk buffer (similarly to the output buffers controlled by `log-fifo-size()`). The hard disk space is not pre-allocated, so ensure that there is always enough free space to store the disk buffers even when the disk buffers are full.

If AxoSyslog 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 in-memory queues. After the restart, AxoSyslog sends the saved messages to the destination. In other words, the disk buffer is persistent. The disk buffer is also resistant to AxoSyslog crashes.

The AxoSyslog 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`, AxoSyslog handles outgoing messages the following way:



- *Output queue*: In-memory queue. If there is space left in it, AxoSyslog puts the message into this queue first . Messages stored here are processed faster, because AxoSyslog can skip writing to, and reading from the disk, as well as serializing or deserializing the message, saving I/O and processor time as a result. The contents of the in-memory output queue are persisted to the disk-buffer file during AxoSyslog reload, restart or stop, but they cannot be persisted if in the event of power failures, or if AxoSyslog crashes. By default, the output queue can hold 1000 messages (you can adjust this number using the `quot-size()` option).
- *Disk-buffer file*: Disk queue. If there is no space left in the output queue, the message is stored on the disk-buffer file. Messages stored here are persisted on the disk, even in case of power failures or if AxoSyslog crashes. Using the disk-buffer file takes considerable amount of disk I/O and processor time. The size of this queue can be set with the `capacity-bytes()` option.
- *Overflow queue*: In-memory queue. This queue is used to trigger flow-control if it is set. The contents of the in-memory overflow queue are persisted to the disk-buffer file in case of AxoSyslog reload, restart or stop, but they are not persisted in case of power failures or if AxoSyslog crashes. Setting the size of the overflow queue can be done with the `flow-control-window-size()` option.

Warning

Hazard of data loss!

In case of normal disk-buffers, the messages stored in the output queue and the overflow queue can be lost in case of power failures or if AxoSyslog crashes.

Note

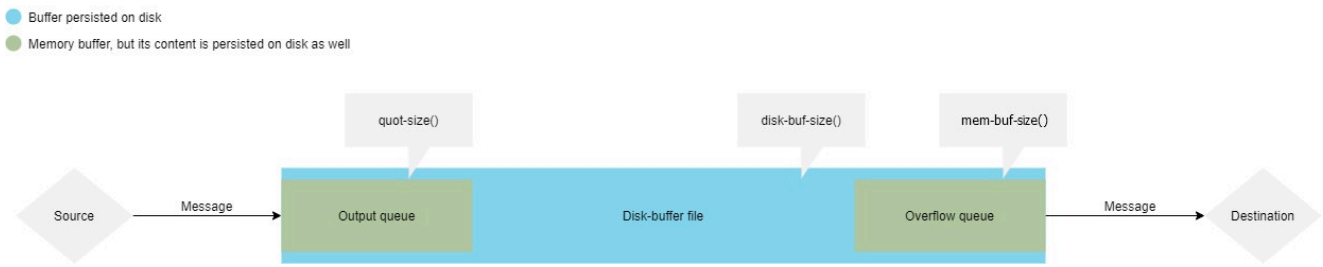
Using disk buffer can significantly decrease performance.

Message handling and reliable disk-based buffering

When you use disk-based buffering, and the `reliable()` option is set to `yes`, AxoSyslog handles outgoing messages the following way.

The `flow-control-window-bytes()` option determines when flow-control is triggered. After the size of the disk-buffer file reaches (`capacity-bytes()` minus `flow-control-window-bytes()`), messages are written into both the disk-buffer file and the overflow queue, indicating that flow-control needs to slow down the message source. These messages are not taken out from the control window (governed by `log-iw-size()`), causing the control window to fill up.

If the control window is full, the flow-control completely stops reading incoming messages from the source. (As a result, `flow-control-window-bytes()` must be at least as large as `log-iw-size()` times the average message size.)



- *Output queue*: In-memory and disk queue. If there is space left in it, AxoSyslog puts the message into this queue first. In case of reliable disk-buffer, in addition to storing the message in memory, it is stored directly in the disk-buffer file as well for safety reasons (see the next point). Messages stored here are processed faster, because AxoSyslog can skip reading from the disk, and deserializing the message, saving I/O and processor time. By default, the output queue can hold 1000 messages (you can adjust it using the `quot-size()` option).
- *Disk-buffer file*: Disk queue. If there is no space left in the output queue, the message is stored on the disk-buffer file. Messages stored here are persisted on the disk, and survive AxoSyslog crash or power failure. Using the disk-buffer file takes considerable amount of disk I/O and processor time. The size of this queue can be set with the `capacity-bytes()` option.
- *Overflow queue*: In-memory and disk queue. This queue is used to trigger flow-control if it is set. Similarly to the output queue, in case of reliable disk-buffer in addition to storing the message in memory, it is stored directly in the disk-buffer file as well for safety reasons. Setting the size of the overflow queue can be done with the `flow-control-window-bytes()` option.

9.3.1 Enabling reliable disk-based buffering

Every destination driver supports the `disk-buffer()` option. The `network()`, `syslog()`, `tcp()`, and `tcp6()` destination drivers can also use the disk-buffer option, except when using the `udp` transport method.

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 AxoSyslog 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: Using reliable disk-based buffering

```
destination d_BSD {
     network("127.0.0.1"
         port(3333)
         disk-buffer(
             flow-control-window-bytes(10000)
             capacity-bytes(2000000)
             reliable(yes)
         )
     );
 };
```

For details on the differences between normal and reliable disk-based buffering, see also About disk queue files.

9.3.2 Enabling normal disk-based buffering

Every destination driver supports the `disk-buffer()` option. The `network()`, `syslog()`, `tcp()`, and `tcp6()` destination drivers can also use the disk-buffer option, except when using the `udp` transport method.

If the `reliable()` option is not set, by default a normal disk-buffer is created. To explicitly enable the normal disk-buffer option, use the `disk-buffer(reliable(no))` parameter in the destination. Use the normal disk-buffer option if you want a solution that is faster than the reliable disk-buffer option. In this case, the process will be less reliable and it is possible to lose logs in case of AxoSyslog crash. The filename of the normal disk-buffer file is the following: `<syslog-ng path>/var/syslog-ng-00000.qf`.

Example: Using normal disk-based buffering

When using the disk-buffer plugin:

```
destination d_BSD {
     network("127.0.0.1"
         port(3333)
         disk-buffer(
             flow-control-window-size(10000)
             capacity-bytes(2000000)
             reliable(no)
         )
     );
 };
```

For details on the differences between normal and reliable disk-based buffering, see also About disk queue files.

9.3.3 How to get information about disk-buffer files

Purpose

This section describes how to get information about disk-buffer files used in AxoSyslog.

> **Note**
>
> **While reading this section, consider that the default installation path used in the commands and AxoSyslog files is `/opt/syslog-ng`.**

9.3.3.1 Information about disk-buffer files

This section describes information about disk-buffer files used in AxoSyslog.

The following list contains information about how disk-buffer files are used in AxoSyslog:

- You can configure `disk-buffer()` for a remote destination in the `destination()` statement.

  For more information about an example of configuring `disk-buffer()` for a remote destination in the `destination()` statement, see disk-buffer() .

- By default, AxoSyslog creates disk-buffer files under `/opt/syslog-ng/var` directory, unless `dir()` option is set in `disk-buffer()` .

- The filenames are generated automatically by AxoSyslog with the extensions `.qf` for a normal disk-buffer and `.rqf` for a reliable disk-buffer.

- The disk-buffer file stores processed log messages in the format in which they would have been sent out to the destination, but doesn't store information about the destination.

9.3.3.2 Getting the status information of disk-buffer files

This section describes how to get status information of the disk-buffer files used in AxoSyslog.

Command syntax

The basic command syntax for getting the status information of the disk-buffer files used in AxoSyslog looks like the following:

```
/opt/syslog-ng/bin/dqtool info <DISK-BUFFER_FILE>
```

Example commands and outputs

- Empty, normal disk-buffer file

  ```
  /opt/syslog-ng/bin/dqtool info /opt/syslog-ng/var/syslog-ng-
  00000.qf/var/lib/syslog-ng/syslog-ng-00000.qf
  ```

  Example output:

  ```
  Disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-
  00000.qf/var/lib/syslog-ng/syslog-ng-00000.qf', number_of_messages='0'
  ```

- Non-empty, reliable disk-buffer queue file

  ```
  /opt/syslog-ng/bin/dqtool info /opt/syslog-ng/var/syslog-ng-00000.rqf
  ```

  ```
  Reliable disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-
  00000.rqf', number_of_messages='10'
  ```

List disk-buffer state in the default directory

You can use the following one-liner command to get the state of all disk-buffer files that are stored in the default directory:

```
for qfile in /opt/syslog-ng/var/*.?(r)qf ; do /opt/syslog-ng/bin/dqtool info $qfile 2>&1 ; done
```

9.3.3.3 Getting the list of disk-buffer files

Purpose

This section describes getting the list of disk-buffer files used in AxoSyslog.

The AxoSyslog application stores information (namely, the `IP:PORT` or `DNS:PORT` of the destinations, and the name of the disk-buffer file) about disk-buffer files in its persist file.

Example: command for listing the disk-buffer files in use

The following command will list the disk-buffer files in use:

```
   /opt/syslog-ng/bin/persist-tool dump /opt/syslog-ng/var/syslog-
ng.persist/var/lib/syslog-ng/syslog-ng.persist | awk -F '["=]' '/(qfile\(|\.queue)/
{ gsub(/[ \t]+/, "", $5); gsub(/^[0-9A-Fa-f]{8}/, "", $5); "echo "$5"|xxd -r -p"|&
getline QUEUE; printf("%s ==> %s\n",$1,QUEUE)}'
```

The example output will look like the following:

```
   afsocket_dd_qfile(stream,10.21.10.20:601)  ==> /opt/syslog-ng/var/syslog-ng-
00000.rqf
```

> **Note**
>
> If you receive the following error message instead of the example output, install a vim-common package on your system:
>
> ```
>    xxd: command not found
> ```

9.3.3.4 Printing the content of disk-buffer files

Purpose

This section describes printing the content of the disk-buffer files used in AxoSyslog.

Command syntax

The command syntax for printing the content of the disk-buffer files used in AxoSyslog looks like the following:

```
/opt/syslog-ng/bin/dqtool cat DISK-BUFFER_FILE
```

Short example output for printed content

Example: short output that shows the printed content of the disk-buffer files used in AxoSyslog

The following short output example shows the printed content of the disk-buffer files used in AxoSyslog:

```
/opt/syslog-ng/bin/dqtool cat /opt/syslog-ng/var/syslog-ng-00000.rqf

 Reliable disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-
00000.rqf', queue_length='2952', size='-437712'
 Jul 31 12:33:48.226 10.21.10.10 <382019-07-31T12:33:36 localhost
prg00000[1234]: seq: 0000000838, thread: 0000, runid: 1564569216, stamp: 2019-07-
31T12:33:36
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPAD
DPADDPADD
 ...
```

9.3.3.5 Orphan disk-buffer files

Purpose

This section describes orphan disk-buffer files used in AxoSyslog.

Orphan disk-buffer files

In certain situations (for example, after modifying the disk-buffer configuration or losing the persist information), AxoSyslog creates a new disk-buffer file instead of using the already existing one. In these situations, the already existing disk-buffer file becomes a so-called orphan disk-buffer file.

> **Note**
>
> **The AxoSyslog application does not store messages in orphan disk-buffer files or forward the messages stored in the disk-buffer file.**

Discovering the new disk-buffer files (orphan disk-buffer files)

To discover orphan disk-buffer files, get the list of disk-buffer files from the persist file, then compare the list with the contents of the disk-buffer files' saving directory.

For more information about how you can get the list of disk-buffer files from the persist file, see Getting the list of disk-buffer files).

Example: difference between the list of disk-buffer files from the persist file and the content of the disk-buffer files' saving directory

The following examples show the difference between the list of disk-buffer files from the persist file and the content of the disk-buffer files' saving directory.

Disk-buffer file list from persist file:

```
   afsocket_dd_qfile(stream,10.21.10.112:514) = { "queue_file": "/opt/syslog-
ng/var/syslog-ng-00001.rqf" }
```

Disk-buffer files' saving directory content:

```
   # ls -l /opt/syslog-ng/var//var/lib/syslog-ng/*qf
    -rw------- 1 root root 2986780 Jul 31 12:30 /opt/syslog-ng/var/syslog-ng-
00000.qf/var/lib/syslog-ng/syslog-ng-00000.qf
    -rw------- 1 root root 2000080 Jul 31 12:31 /opt/syslog-ng/var/syslog-ng-
00000.rqf
    -rw------- 1 root root    4096 Aug  1 11:09 /opt/syslog-ng/var/syslog-ng-
00001.rqf
```

The disk-buffer files `syslog-ng-00000.qf` and `syslog-ng-00000.rqf` don't exist in the persist file. These two files are the orphan disk-buffer files.

For more information about orphan disk-buffer files and how to process the messages in orphan disk-buffer files using a separate AxoSyslog instance, see Process orphan disk-buffer files.

9.3.3.6 Process orphan disk-buffer files

Purpose

When AxoSyslog creates orphan disk-buffer files, you can start a separate AxoSyslog instance parallel to the AxoSyslog instance already running, and use the following procedure to process the messages in the orphan disk-buffer file.

Orphan disk-buffer files

In certain situations (for example, after modifying the disk-buffer configuration or losing the persist information), AxoSyslog creates a new disk-buffer file instead of using the already existing one. In these situations, the already existing disk-buffer file becomes a so-called orphan disk-buffer file.

> **Note**
>
> **The AxoSyslog application does not store messages in orphan disk-buffer files or forward the messages stored in the disk-buffer file.**

Steps

> **Warning**
>
> Before starting a separate AxoSyslog instance to process the messages from the orphan disk-buffer file, consider the following:
>
> - During the resolution process, a separate AxoSyslog instance will be started with its temporary files beside the AxoSyslog instance already running.
> - An incorrect startup command and incorrect configurations may cause issues for the AxoSyslog instance already running.
> - The disk-buffer file stores processed log messages in the format in which they would have been sent out to the destination.
> - The disk-buffer file doesn't store information about the destination.

1. Identify the orphan disk-buffer files and make a record of them. For more information, see How to get information about disk-buffer files.

   It is important to know the type of the disk-buffer file. Disk-buffer file types can be normal (`.qf`) or reliable (`.rqf`). In the examples during this process, the `/opt/syslog-ng/var/syslog-ng-00005.rqf` orphan reliable disk-buffer file is used.

2. Determine the destination of the logs. The content of the disk-buffer may help you determine the logs' destination. For more information, see How to get information about disk-buffer files.

   In the examples the destination `10.21.10.20` is used with the standard `network()` port `514` .

3. Create a directory for the temporary instance. In the examples during this process, the `/tmp/qdisk` directory is used.

```
mkdir /tmp/qdisk
```

> **Warning**
>
> **Make sure that there is sufficient disk space in the directory. The minimum recommended disk space in the directory is equal to the size of the orphan disk-buffer file.**

4. Create the configuration file `/tmp/qdisk/qdisk.conf` for the temporary instance with the following content.

```
@version:4.29
@include "scl.conf"

options {
    keep-hostname(yes);
    keep-timestamp(yes);
};

destination d_destination {
#    ADD YOUR DESTINATION HERE

};

log {
    destination(d_destination);
};
```

5. Add your destination statement with `disk-buffer()` to the configuration file. You can copy the destination statement from your running AxoSyslog configuration.

> **Warning**
>
> **Add the `dir()` option and set the disk-buffer file's destination directory to the temporary directory (that is, `/tmp/qdisk`) in your destination statement.**

6. Start the temporary AxoSyslog instance in the foreground.

```
syslog-ng -Fe -f /tmp/qdisk/qdisk.conf -R /tmp/qdisk/qdisk.persist -c
/tmp/qdisk/qdisk.ctl
```

The AxoSyslog application will log to the console, so you will see any potential error that may occur during startup.

The following example output displays that an empty disk-buffer file has been created and the connection to the remote destination has been established.

```
Follow-mode file source not found, deferring open;
filename='/no_such_file_or.dir'
Reliable disk-buffer state saved; filename='/tmp/qdisk/syslog-ng-00000.rqf',
qdisk_length='0'
No server license found, running in client mode;
syslog-ng starting up; version='7.0.20', cfg-
fingerprint='eaa03b9efb88b87d7c1b0ce7efd042ed8ac0c013', cfg-nonce-ndx='0', cfg-
signature='c0327a7f7e6418ce0399a75089377dfb662bb072'
FIPS information; FIPS-mode='disabled'
Syslog connection established; fd='7', server='AF_INET(10.21.10.20:514)',
local='AF_INET(0.0.0.0:0)'
```

7. To stop AxoSyslog, press `CTRL+C`.

8. Overwrite the empty disk-buffer file with the orphan disk-buffer file.

```
mv /opt/syslog-ng/var/syslog-ng-00005.rqf /tmp/qdisk/syslog-ng-00000.rqf
```

9. Start AxoSyslog using the command used in Start the temporary AxoSyslog instance in the foreground step.

```
syslog-ng -Fe -f /tmp/qdisk/qdisk.conf -R /tmp/qdisk/qdisk.persist -c
/tmp/qdisk/qdisk.ctl
```

10. Open another terminal and check the progress by using one of the following methods.

    - Checking the number of stored logs in the disk-buffer (that is, the last number from the output).

      ```
      /opt/syslog-ng/sbin/syslog-ng-ctl stats -c /tmp/qdisk/qdisk.ctl | grep
      'dst.*queued'
      ```

    - Checking the status of the disk-buffer file.

      ```
      /opt/syslog-ng/bin/dqtool info /tmp/qdisk/syslog-ng-00000.rqf
      ```

      An empty disk-buffer file will look similar to this:

      ```
      Reliable disk-buffer state loaded; filename='/tmp/qdisk/syslog-ng-
      00000.rqf', queue_length='0', size='0'
      ```

11. Press `CTRL+C` to stop AxoSyslog.
12. Check the state of the orphan disk-buffer file. For more information, see How to get information about disk-buffer files.
13. If you have more than one orphan disk-buffer file, repeat the steps following the AxoSyslog stop (that is, the steps beginning from overwriting the empty disk-buffer file with the orphan disk-buffer file) for each orphan disk-buffer file.
14. Remove the temporary directory.

```
rm -rf /tmp/qdisk
```

9.3.4 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 AxoSyslog 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: Using memory buffering

```
destination d_BSD {
     network("127.0.0.1"
         port(3333)
         log-fifo-size(10000)
     );
 };
```

9.3.5 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 AxoSyslog crashes. Of course, using the `reliable(yes)` option introduces a significant performance penalty as well.

Both reliable and normal disk-buffers employ an in-memory output queue (set in `quot-size()`) and an in-memory overflow queue (set in `flow-control-window-bytes()` for reliable disk-buffers, or `flow-control-window-size()` for normal disk-buffers). The difference between reliable and normal disk-buffers is that when the reliable disk-buffer uses one of its in-memory queues, it also stores the message on the disk, whereas the normal disk-buffer stores the message only in memory. The normal disk-buffer only uses the disk if the in-memory output buffer is filled up completely. This approach has better performance (due to fewer disk I/O operations), but also carries the risk of losing a maximum of `quot-size()` plus `flow-control-window-size()` number of messages in case of an unexpected power failure or application crash.

Size of the queue files

Disk queue files grow. Each may take up to `capacity-bytes()` bytes on the disk. Due to the nature of reliable queue files, all the messages traversing the queue are written to disk, constantly increasing the size of the queue file.

The disk-buffer file's size should be considered as the configured `capacity-bytes()` at any point of time, even if it does not have messages in it. Truncating the disk-buffer file can slow down disk I/O operations, so AxoSyslog does not always truncate the file when it would be possible (see the `truncate-size-ratio()` option). If a large disk-buffer file is not desirable, you should set the `capacity-bytes()` option to a smaller value. Note that AxoSyslog version 4.0 and later doesn't truncate disk-buffer files by default.

Starting with AxoSyslog version 4.0, you can preallocate disk-buffer files .

> **Note**
>
> **The disk-buffer file's size does not strictly correlate to the number of stored messages. If you want to get information about the disk-buffer, use `dqtool` (for more information, see Getting the status information of disk-buffer files).**

> **Note**
>
> **If a queue file becomes corrupt, AxoSyslog 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.**

Preallocating disk-buffer files

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

9.4 Filters

> ### FilterX or classic filters, parsers, and rewrite rules?
>
> **FilterX replaces filters, parsers, and rewrite rules with a single typed language that can also handle nested JSON and other complex data. You don't have to migrate: the classic blocks keep working, and you can use both in the same log path. To convert your existing configuration, see Update to FilterX.**

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 AxoSyslog.

9.4.1 Using filters

Filters perform log routing: 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.conf` 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);
};
```

9.4.2 Combining filters with boolean operators

When a log statement includes multiple filter statements, AxoSyslog 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), or optionally to the content of any other macro. The `template()` parameter of the `match` function can be used to run a filter against a combination of macros. For details on using the `match()` filter function, see match().

> **Note**
>
> **Filters are often used together with log path flags. For details, see Log path flags.**

9.4.3 Comparing macro values in filters

In AxoSyslog you can 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. For details on macros and templates, see Customize message format using macros and templates.

Use the following syntax to compare macro values or templates.

```
filter <filter-id>
    {"<macro-or-template>" operator "<value-or-macro-or-template>"};
```

String and numerical comparison

AxoSyslog versions earlier than 4.0 used separate operators for string comparisons (for example, `eq`). In version 4.0 and later, you can simply use the mathematical symbols as operators (like `==, !=, >=`), and AxoSyslog automatically determines how to compare the arguments from their type. The logic behind that is similar to JavaScript:

- If both sides of the comparisons are strings, then the comparison is string.
- If one of the arguments is numeric, then the comparison is numeric.
- Literal numbers (numbers not enclosed in quotes) are numeric.
- You can explicitly type-cast an argument into a number.

For example:

- `if ("${.apache.httpversion}" == 1.0)`

  The right side of the == operator is 1.0, which is a floating point literal, so the comparison is numeric.

- `if (double("${.apache.httpversion}") == "1.0")`

  The left side is explicitly type cast into double, the right side is string (because of the quotes), so the comparison is numeric.

- `if ("${.apache.request}" == "/wp-admin/login.php")`

  The left side is not type-cast, so it's a string, the right side is a string, so the comparison is string.

Note: You can still use the old string operators if you want to, they are available for backwards compatibility.

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}" == "${HOST}"};
```

The following example selects messages with priority level higher than 5.

```
filter f_level {"${LEVEL_NUM}" > 5};
```

Make sure to:

- Enclose macros and templates in double-quotes.
- Use the `$` character before macros.

Note that:

- You can use type casting anywhere where you can use templates to apply a type to the result of the template expansion.
- Using comparator operators can be equivalent to using filter functions, but is somewhat slower. For example, using `"${HOST}" == "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.

Comparison operators

The following numerical and string comparison operators are available.

Numerical or string 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

9.4.4 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, AxoSyslog 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 wildcards

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));};
```

To filter for special control characters like the carriage return (CR), use the `\\r` escape prefix in AxoSyslog version 3.0 and 3.1. In AxoSyslog 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"));};
```

9.4.5 Tagging messages

You can label the messages with custom tags. Tags are simple labels, identified by their names, which must be unique. Currently AxoSyslog can tag a message at the following places:

- At the source when the message is received.
  - AxoSyslog automatically adds the `.source.<id_of_the_source_statement>` tag to every incoming message.
  - You can add custom tag using the `tags()` option of the source.
  - AxoSyslog 4.7 and newer automatically adds the following tags if it encounters errors when parsing syslog messages.
    - `message.utf8_sanitized` : The message is not valid UTF-8.
    - `syslog.missing_timestamp` : The message has no timestamp.
    - `syslog.invalid_hostname` : The hostname field doesn't seem to be valid, for example, it contains invalid characters. For details on the valid characters, see the `check-hostname()` global option .
    - `syslog.missing_pri` : The priority (PRI) field is missing from the message.
    - `syslog.unexpected_framing` : An octet count was found in front of the message, suggested invalid framing.
    - `syslog.rfc3164_missing_header` : The date and the host are missing from an RFC3164-formatted message - practically that's the entire header of RFC3164-formatted messages.
    - `syslog.rfc5424_unquoted_sdata_value` : An RFC5424 message contains an incorrectly quoted SDATA field.
    - `message.parse_error` : Some other parsing error occurred.
- 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 pattern database format.
- Tags can be also added and deleted using rewrite rules. For details, see Add and delete tags.

You can use the `tags()` filter to select only specific messages.

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

macro is available only in AxoSyslog 3.1.1 and later.

For an example on tagging, see Example: Adding tags and filtering messages with tags .

9.4.6 Filter functions

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

Filter functions available in AxoSyslog

Name	Description
facility()	Filter messages based on the sending facility.
filter()	Call another filter function.
host()	Filter messages based on the sending host.
in-list()	File-based whitelisting and blacklisting.
level() or priority()	Filter messages based on their priority.
match()	Use a regular expression to filter messages based on a specified header or content field.
message()	Use a regular expression to filter messages based on their content.
netmask() or netmask6()	Filter messages based on the IP address of the sending host.
program()	Filter messages based on the sending application.
rate-limit()	Limits messages rate based on arbitrary keys in each message.
source()	Select messages of the specified AxoSyslog source statement.
tags()	Select messages having the specified tag.

9.4.6.1 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 AxoSyslog application recognizes the following facilities: (Note that some of these facilities are available only on specific platforms.)

The `facility()` filter recognizes the following syslog facilities:

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)

9.4.6.2 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) }
```

9.4.6.3 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, AxoSyslog 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") };
```

9.4.6.4 in-list()

Synopsis: `in-list("</path/to/file.list>", value("<field-to-filter>"))`

Description: Matches the value of the specified field to a list stored in a file, allowing you to do simple, file-based black- and whitelisting. The file must be a plain-text file, containing one entry per line.

The AxoSyslog 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 AxoSyslog for the changes to take effect.
- Make sure that there is an empty line in the file after the last entry. If the last entry is the last line, it will be excluded from the list.

Available in AxoSyslog 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. Note the empty line after the last entry.

```
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")); };
```

9.4.6.5 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` .

9.4.6.6 match()

Synopsis: `match(regexp)` , `match(regexp value("MACRO"))` , `match(regexp template("MACROS"))`

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 AxoSyslog 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.

Starting with version 3.22, the `match()` filter can work on templates as well. This means that you can a match against an expression combined of macros, instead of a single macro. Note that when using a template, you must reference macros with the $ sign (unlike when using the `value()` parameter). For example:

```
match("^my-regular-expression" template("${HOST}|${PROGRAM}${PID}|${MESSAGE}"));
```

Using a template with a single macro is equivalent with using the `value()` parameter. For example, the following two lines are equivalent:

```
match("^my-regular-expression" value("MESSAGE"));
match("^my-regular-expression" template("${MESSAGE}"));
```

9.4.6.7 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.

9.4.6.8 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 AxoSyslog), 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().

9.4.6.9 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 AxoSyslog), 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, AxoSyslog will ignore the bits of the address after the prefix. To filter IPv4 addresses, see netmask().

The `netmask6()` filter is available in AxoSyslog 3.7 and later.

> **Warning**
>
> **If the IP address is not syntactically correct, the filter will never match. The AxoSyslog application currently does not send a warning for such configuration errors.**

9.4.6.10 program()

Synopsis: program(regexp)

Description: Match messages by using a regular expression against the program name field of log messages.

9.4.6.11 rate-limit()

Synopsis: `rate-limit(key($HOST) rate(5000))`

Description: Limits messages rate based on arbitrary keys in each message. The key will be resolved using the `key()` option. Each resolution will be allowed to have the number of messages each second, set by the `rate()` option. For example if `key($HOST)` and `rate(5000)` are set, and there are 2 hosts sending messages to AxoSyslog, a total of `10000` messages will be allowed by the `rate-limit()` filter, `5000` from the first and `5000` from the second host. If `key()` was not set instead, then `5000` messages would be allowed each second, regardless of their content.

> **Note**
>
> **In AxoSyslog version 3.35 the `rate-limit()` filter was called `throttle()`. In AxoSyslog version 3.36 it got renamed to `rate-limit()`, but `throttle()` is still available for backward compatibility.**

> **Note**
>
> **Like every other filter, messages unmatched (outside of the rate limit) by the `rate-limit()` filter are dropped by default. Also, as every filter can be used in channels or if conditions, the messages unmatched can be caught and handled, like sent to a different destination, and so on.**

Example: Using the rate-limit() filter

The following example depicts the scenario described in the description part of this section.

```
filter f_rate_limit {
                rate-limit(
                key("$HOST")
                rate(5000)
                );
            };
```

9.4.6.11.1 Options of rate-limit() filter

The rate-limit() filter has the following options.

key()

Type:	template
Default:	empty string

Description: The resolved template, that will be used to create unique rate-limit token buckets. In AxoSyslog version 4.4 and earlier, the name of this option was `template()` .

rate()

Type:	number
Default:	N/A
Mandatory:	yes

Description: The number of messages for each unique macro resolution, that will be let through (matched) by the filter each second.

9.4.6.12 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.

9.4.6.13 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 AxoSyslog 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 AxoSyslog 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");};
> ```

9.5 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); };
```

10 FilterX

What FilterX is, when to use it instead of classic filters, parsers, and rewrite rules, a minimal example, and a map of the FilterX pages.

FilterX is the AxoSyslog language to filter, parse, and rewrite log messages. It replaces filters , parsers , and rewrite rules with a single language that has typed variables, and handles nested data structures like JSON.

A FilterX block is a list of statements. A message passes the block only if every statement is true for the message. For example, you can select messages from a specific host, parse the message into fields, then change or delete some of the fields.

To try FilterX in a few minutes, see Get started with FilterX.

FilterX or classic filters, parsers, and rewrite rules?

Use FilterX for new configurations, and when you process structured data like JSON, key=value pairs, or OpenTelemetry records. FilterX is faster than the classic blocks, and handles multi-level, typed objects.

You don't have to migrate existing configurations: the classic blocks keep working. You can use FilterX and classic blocks in the same log path, for example, a parser before a FilterX block. To convert existing filters and rewrite rules, see Update to FilterX.

Define a filterx block

You can define `filterx` blocks inline in your log statements. (If you want to reuse `filterx` blocks, Reuse FilterX blocks.)

For example, the following FilterX statement selects the messages that contain the word `deny` and come from the host `example` .

```
log {
    source(s1);
    filterx {
        ${HOST} == "example";
        ${MESSAGE} =~ "deny";
    };
    destination(d1);
};
```

You can use `filterx` blocks together with other blocks in a log path, for example, use a parser before/after the `filterx` block if needed.

What's in this section

- FilterX language basics: statements, variables, types, and scope.
- FilterX use cases and examples: common tasks and a FilterX iptables parser.
- Comparing, boolean, conditional, and string search pages: how to write conditions.
- Parsing data in FilterX and Format data: parse and format CSV, key=value, JSON, XML, CEF, LEEF, and other formats.
- FilterX operator reference and FilterX function reference: the complete A–Z references.

Operators and functions

FilterX has the following types of operators:

- Comparison: `==`, `===`, `<`, and others.
- Boolean: `and`, `or`, `not`.
- Regular expression match: `=~`, `!~`.
- Arithmetic and string: `+`, `+=`, `..` (slicing).
- Null handling: `??`, `=??`.

For the complete list, see FilterX operator reference.

FilterX has built-in functions to parse, format, and convert data. Other functions handle strings, dates, timezones, hashes, and metrics. For the complete list, see FilterX function reference.

Next steps

- Get started with FilterX
- FilterX language basics
- FilterX function reference
- FilterX operator reference
- Update to FilterX

10.1 Get started with FilterX

Build your first FilterX block: filter, parse a key=value message, set a field, and check the output in a file.

In this tutorial, you build a FilterX block that processes iptables log messages. You learn how to filter messages, parse key=value pairs, modify a field, and send the result to a destination.

Prerequisites

- Docker. For details, see Install AxoSyslog with Docker. To use Podman instead, see Install AxoSyslog with Podman and replace all `docker` commands with `podman`.
- Basic knowledge of AxoSyslog configuration: sources, destinations, and log paths.

Sample messages

You use two sample messages in this tutorial. The first one is an iptables message from the kernel. It's an RFC3164-formatted message with space-separated key=value pairs:

```
Dec 08 12:00:00 hostname.example kernel: IN=eth0 OUT=
MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00 SRC=192.0.2.2 DST=192.168.0.1 LEN=40
TOS=0x00 PREC=0x00 TTL=232 ID=12345 PROTO=TCP SPT=54321 DPT=22 WINDOW=1023 RES=0x00
SYN URGP=0
```

The second one is an sshd message. Your FilterX block must not let it through:

```
Dec 08 12:00:01 hostname.example sshd[1234]: Accepted publickey for alice from
192.0.2.2 port 54321 ssh2
```

Step 1: Start AxoSyslog without FilterX

First, you run a minimal configuration that reads messages from a file and writes them to another file.

1. Create an empty directory and an empty input file:

```
mkdir filterx-tutorial
cd filterx-tutorial
touch input.log
```

2. Create a file called `filterx-tutorial.conf` with the following content:

```
@version: current

source s_tutorial {
    file("/tutorial/input.log" follow-freq(1));
};

destination d_tutorial {
    file("/tutorial/output.log" template("${PROGRAM}: ${MESSAGE}\n"));
};

log {
    source(s_tutorial);
    destination(d_tutorial);
};
```

3. Start AxoSyslog in a container. The container mounts the current directory as `/tutorial`, and keeps running in the background.

```
docker run -d --name filterx-tutorial -v "$PWD:/tutorial"
ghcr.io/axoflow/axosyslog:latest -f /tutorial/filterx-tutorial.conf
```

4. Append the two sample messages to the input file:

```
echo 'Dec 08 12:00:00 hostname.example kernel: IN=eth0 OUT=
MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00 SRC=192.0.2.2 DST=192.168.0.1
LEN=40 TOS=0x00 PREC=0x00 TTL=232 ID=12345 PROTO=TCP SPT=54321 DPT=22
WINDOW=1023 RES=0x00 SYN URGP=0' >> input.log
echo 'Dec 08 12:00:01 hostname.example sshd[1234]: Accepted publickey for alice
from 192.0.2.2 port 54321 ssh2' >> input.log
```

5. Check the output file:

```
tail -n 2 output.log
```

You should see both messages:

```
kernel: IN=eth0 OUT= MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00
SRC=192.0.2.2 DST=192.168.0.1 LEN=40 TOS=0x00 PREC=0x00 TTL=232 ID=12345
PROTO=TCP SPT=54321 DPT=22 WINDOW=1023 RES=0x00 SYN URGP=0
sshd: Accepted publickey for alice from 192.0.2.2 port 54321 ssh2
```

Keep the container running. In the next steps, you edit `filterx-tutorial.conf`, then apply the changes with the following command:

```
docker exec filterx-tutorial syslog-ng-ctl reload
```

You should see:

```
Config reload successful
```

Step 2: Filter the messages

Now you add a FilterX block that lets only iptables messages through.

1. Replace the log path in `filterx-tutorial.conf` with the following:

```
log {
    source(s_tutorial);
    filterx {
        ${PROGRAM} == "kernel";
        ${MESSAGE} =~ "PROTO=";
    };
    destination(d_tutorial);
};
```

A message passes the FilterX block only if every statement in the block is true. For details, see Truthy and falsy values . The `=~` operator checks if the message matches a regular expression. For details, see FilterX operator reference.

2. Reload the configuration:

```
docker exec filterx-tutorial syslog-ng-ctl reload
```

3. Append the two sample messages again:

```
echo 'Dec 08 12:00:00 hostname.example kernel: IN=eth0 OUT=
MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00 SRC=192.0.2.2 DST=192.168.0.1
LEN=40 TOS=0x00 PREC=0x00 TTL=232 ID=12345 PROTO=TCP SPT=54321 DPT=22
WINDOW=1023 RES=0x00 SYN URGP=0' >> input.log
echo 'Dec 08 12:00:01 hostname.example sshd[1234]: Accepted publickey for alice
from 192.0.2.2 port 54321 ssh2' >> input.log
```

4. Check the output file:

```
tail -n 3 output.log
```

You should see:

```
kernel: IN=eth0 OUT= MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00
SRC=192.0.2.2 DST=192.168.0.1 LEN=40 TOS=0x00 PREC=0x00 TTL=232 ID=12345
PROTO=TCP SPT=54321 DPT=22 WINDOW=1023 RES=0x00 SYN URGP=0
sshd: Accepted publickey for alice from 192.0.2.2 port 54321 ssh2
kernel: IN=eth0 OUT= MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00
SRC=192.0.2.2 DST=192.168.0.1 LEN=40 TOS=0x00 PREC=0x00 TTL=232 ID=12345
PROTO=TCP SPT=54321 DPT=22 WINDOW=1023 RES=0x00 SYN URGP=0
```

The first two lines are from step 1. The last line is new: the kernel message passed the filter, but the sshd message didn't.

Step 3: Parse the key=value pairs

Next, you parse the iptables message into separate fields, and send them as JSON.

1. Add the following lines to the end of the FilterX block, after the two filter statements:

   ```
   iptables = parse_kv(${MESSAGE}, value_separator="=", pair_separator=" ");
   ${MESSAGE} = iptables;
   ```

   - `parse_kv` splits the message into key=value pairs, and stores them in the `iptables` variable.
   - `iptables` is a local FilterX variable, because its name doesn't start with `$`. AxoSyslog doesn't send local variables to the destination, unless you assign them to a name-value pair. For details, see FilterX variables in destinations .
   - Assigning the `iptables` dict to `${MESSAGE}` stores it as JSON, so the destination receives a JSON string. You don't have to format it explicitly. To send it in another format, use a formatting function, like `format_kv` . For details, see Format data.
2. Reload the configuration, and append the two sample messages again, the same way as in Step 2 .
3. Check the output file:

   ```
   tail -n 1 output.log
   ```

   You should see the parsed message as JSON:

   ```
   kernel:
   {"IN":"eth0","OUT":"","MAC":"11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00","SRC":"
   192.0.2.2","DST":"192.168.0.1","LEN":"40","TOS":"0x00","PREC":"0x00","TTL":"232
   ","ID":"12345","PROTO":"TCP","SPT":"54321","DPT":"22","WINDOW":"1023","RES":"0x
   00","URGP":"0"}
   ```

   The word `SYN` isn't in the output, because it isn't a key=value pair.

Step 4: Rename a field

Finally, you rename the `SRC` field to `src_ip`.

1. Add the following lines to the FilterX block, before the `${MESSAGE} = iptables;` line:

   ```
   iptables.src_ip = iptables.SRC;
   unset(iptables.SRC);
   ```

   The first line copies the value of `SRC` to a new `src_ip` field. The `unset` function deletes the `SRC` field.

   The FilterX block now looks like this:

   ```
   filterx {
       ${PROGRAM} == "kernel";
       ${MESSAGE} =~ "PROTO=";

       iptables = parse_kv(${MESSAGE}, value_separator="=", pair_separator=" ");
       iptables.src_ip = iptables.SRC;
       unset(iptables.SRC);
       ${MESSAGE} = iptables;
   };
   ```

2. Reload the configuration, and append the two sample messages again, the same way as in Step 2.

3. Check the output file:

   ```
   tail -n 1 output.log
   ```

   You should see:

```
kernel:
{"IN":"eth0","OUT":"","MAC":"11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00","DST":"
192.168.0.1","LEN":"40","TOS":"0x00","PREC":"0x00","TTL":"232","ID":"12345","PR
OTO":"TCP","SPT":"54321","DPT":"22","WINDOW":"1023","RES":"0x00","URGP":"0","sr
c_ip":"192.0.2.2"}
```

The `SRC` field is gone, and the `src_ip` field is at the end of the JSON object.

What you built

You now have a FilterX block that selects iptables messages, parses them into fields, and renames a field. AxoSyslog sends the result to a file as JSON. Your complete `filterx-tutorial.conf` file looks like this:

```
@version: current

source s_tutorial {
    file("/tutorial/input.log" follow-freq(1));
};

destination d_tutorial {
    file("/tutorial/output.log" template("${PROGRAM}: ${MESSAGE}\n"));
};

log {
    source(s_tutorial);
    filterx {
        ${PROGRAM} == "kernel";
        ${MESSAGE} =~ "PROTO=";

        iptables = parse_kv(${MESSAGE}, value_separator="=", pair_separator=" ");
        iptables.src_ip = iptables.SRC;
        unset(iptables.SRC);
        ${MESSAGE} = iptables;
    };
    destination(d_tutorial);
};
```

Clean up

To stop and remove the container, run:

```
docker rm -f filterx-tutorial
```

Next steps

- Reuse FilterX blocks
- Update to FilterX
- FilterX function reference
- Create an iptables parser

10.2 FilterX language basics

FilterX statements, truthy and falsy values, variable scope and types, assigning and deleting values, and complex types like dicts, lists, and tuples.

This page describes the building blocks of the FilterX language: statements, variables, types, and how FilterX data reaches your destinations. For a hands-on introduction, see Get started with FilterX.

Truthy and falsy values

FilterX blocks consist of a list of FilterX statements, each statement evaluates either to *truthy* or *falsy*. If a message matches all FilterX statements, it passes through the FilterX block to the next element of the log path, for example, the destination.

- Truthy values are:
  - Complex values (for example, a datetime object),
  - non-empty lists and objects,
  - non-empty strings,
  - non-zero numbers,
  - the `true` boolean object.
- Falsy values are:
  - empty strings,
  - the `false` value,
  - the `0` value,
  - `null`,

Statements that result in an error (for example, if a comparison cannot be evaluated because of type error, or a field or a dictionary referenced in the statement doesn't exist or is unset) are also treated as falsy. To find and fix such errors, see Troubleshoot FilterX.

FilterX statements

A FilterX block contains one or more FilterX statements. The order of the statements is important, as they are processed sequentially. If any of the statements is falsy (or results in an error), AxoSyslog drops the message from that log path.

FilterX statements can be one of the following:

- A comparison, for example, `${HOST} == "my-host";`. This statement is true only for messages where the value of the `${HOST}` field is `my-host`. Such simple comparison statements can be the equivalents of traditional filter functions.
- A value assignment for a name-value pair or a local variable, for example, `${my-field} = "bar";`. The left-side variable automatically gets the type of the right-hand expression. Assigning the false value to a variable (`${my-field} = false;`) is a valid statement that doesn't automatically cause the FilterX block to return as false.
- Existence of a variable of field. For example, the `${HOST};` expression is true only if the `${HOST}` macro exists and isn't empty.
- A conditional statement (`if (expr) { ... } elif (expr) {} else { ... };`) which allows you to evaluate complex decision trees. Starting with version 4.10, you can also use switch-case expressions.
- A declaration of a pipeline variable, for example, `declare my_pipeline_variable = "something";`.
- A FilterX action. This can be one of the following:
  - `drop;`: Intentionally drop the message. This means that the message was successfully processed, but discarded. Processing the dropped message stops at the `drop` statement, subsequent sections or other branches of the FilterX block won't process the message. For example, you can use this to discard unneeded messages, like debug logs. Available in AxoSyslog 4.9 and later.
  - `done;`: Return truthy and don't execute the rest of the FilterX block, returns with true. This is an early return that you can use to avoid unnecessary processing, for example, when the message matches an early classification in the block. Available in AxoSyslog 4.9 and later.

> **Note**
>
> - The `true;` and `false;` literals are also valid as statements. They can be useful in complex conditional (if/elif/else) statements.
> - A name-value pair or a variable in itself is also a statement. For example, `${HOST};`. If the name-value pair or variable is empty or doesn't exist, the statement is considered falsy.

When you assign the value of a variable using another variable (for example, `${MESSAGE} = ${HOST};`), AxoSyslog copies the current value of the `${HOST}` variable. If a statement later changes the value of the `${HOST}` field, the `${MESSAGE}` field won't change. For example:

```
filterx {
  ${HOST} = "first-hostname";
  ${MESSAGE} = ${HOST}; # The value of ${MESSAGE} is first-hostname
  ${HOST} = "second-hostname"; # The value of ${MESSAGE} is still first-hostname
};
```

The same is true for complex objects, like JSON, for example:

```
js = json({
    "key": "value",
    "second-key": "another-value"
});

${MESSAGE} = js;

js.third_key = "third-value-not-available-in-MESSAGE";
```

You can use FilterX operators and functions .

Data model and scope

Each FilterX block can access data from the following elements.

- Macros and name-value pairs of the message being processed (for example, `$PROGRAM`). The names of macros and name-value pairs begin with the `$` character. If you define a new variable in a FilterX block and its name begins with the `$` character, it's automatically added to the name-value pairs of the message.

> **Note**
>
> Using curly braces around macro names is not mandatory, and the `"$MESSAGE"` and `"${MESSAGE}"` formats are equivalent. If the name contains only alphanumeric characters and the underscore character, you don't need the curly braces. If it contains any other characters (like a hyphen (`-`) or a dot (`.`)), you need to add the curly braces, therefore it's best to always use curly braces.
>
> Names are case-sensitive, so `"$message"` and `"$MESSAGE"` are not the same.

- Local variables. These have a name that doesn't start with a `$` character, for example, `my_local_variable`. Local variables are available only in the FilterX block they're defined.
- Pipeline variables. These are similar to local variables, but must be declared before first use, for example, `declare my_pipeline_variable=5;`

  Pipeline variables are available in the current and all subsequent FilterX block. They're global in the sense that you can access them from multiple FilterX blocks, but note that they're still attached to the particular message that is processed, so the values of pipeline variables aren't preserved between messages.

  If you don't need to pass the variable to another FilterX block, use local variables, as pipeline variables have a slight performance overhead.

> **Note**
>
> - If you want to pass data between two FilterX blocks of a log statement, use pipeline variables. That has better performance than name-value pairs.
> - Local and pipeline variables aren't available in destination templates. For details, see FilterX variables in destinations .

Variable names

FilterX variable names have more restrictions than generic name-value pair names. They:

- can contain alphanumeric characters and the underscore character (`_`), but **cannot** contain hyphens,
- cannot begin with numbers,
- can begin with underscore.

> **Note**
>
> **Although you can re-use type names and function names as variable names, that's not considered good practice and should be avoided.**

Variable types

Variables can have the following types. All of these types have a matching function that can be used to type cast something into the specific type.

- `boolean`

- `bytes` (to represent binary data)
- `datetime`
- `dict`
- `double`
- `integer`
- `ip` for individual IPv4 or IPv6 addresses.
- `json` and `json_array` for JSON or JSON-like objects.
- `list`
- `metrics_labels`
- `null`
- `otel_array`
- `otel_kvlist`
- `otel_logrecord`
- `otel_resource`
- `otel_scope`
- `protobuf`
- `string`
- `subnet` for IPv4 or IPv6 subnets in CIDR notation.
- `tuple` is a read-only, list-like data type similar to a Python tuple. Available in AxoSyslog 4.27 and later.

Assign values

To assign value to a name-value pair or a variable, use the following syntax:

```
<variable-name> = <value-of-the-variable>;
```

In most cases you can omit the type, and AxoSyslog automatically assigns the type based on the syntax of the value, for example:

- `mystring = "string-value";`
- `myint = 3;`
- `mydouble = 2.5;`
- `myboolean = true;`

When needed, you can explicitly specify the type of the variable, and AxoSyslog attempts to convert the value to the specified type:

```
<variable-name> = <variable-type>(<value-of-the-variable>);
```

For example:

```
filterx {
  ${MESSAGE} = string("Example string message");
};
```

You can also assign the value of other name-value pairs, for example:

```
filterx {
  ${MESSAGE} = ${HOST};
};
```

When processing RFC5424-formatted (IETF-syslog) messages, you can modify the SDATA part of the message as well. The following example sets the sequenceId:

```
filterx {
  ${.SDATA.meta.sequenceId} = 55555;
};
```

> **Note**
>
> **When assigning values to name-value pairs, you cannot modify hard macros .**

Template functions

You can use the traditional template functions of AxoSyslog to access and format name-value pairs. For that you must enclose the template function expression between double-quotes, for example:

```
${MESSAGE} = "$(format-json --subkeys values.)";
```

However, note that template functions cannot access the local and pipeline variables created in FilterX blocks.

Delete values

To delete a value without deleting the object itself (for example, name-value pair), use the `null value`, for example:

```
${MY-NV-PAIR-KEY} = null;
```

To delete the name-value pair (or a key from an object), use the `unset` function:

```
unset(${MY-NV-PAIR-KEY});
unset(${MY-JSON}["key-to-delete"]);
```

To unset every empty field of an object, use the `unset-empties` function:

> **Note**
>
> **You cannot unset hard macros .**

Add two values

The plus operator (`+`) adds two arguments, if possible. (For example, you can't add two datetime values.)

- You can use it to add two numbers (two integers, two double values). If you add a double to an integer, the result is a double.
- Adding two strings concatenates the strings. Note that if you want to have spaces between the added elements, you have to add them manually, like in Python, for example:

```
${MESSAGE} = ${HOST} + " first part of the message," + " second part of the
message" + "\n";
```

- Adding two lists merges the lists. Available in AxoSyslog 4.9 and later.
- Adding two dicts updates the dict with the values of the second operand. For example:

```
x = {"key1": "value1", "key2": "value1"};
y = {"key3": "value1", "key2": "value2"};
${MESSAGE} = x + y; # ${MESSAGE} value is {"key1": "value1", "key3": "value1",
"key2": "value2"};
```

Available in AxoSyslog 4.9 and later.

For other arithmetic operators, see Arithmetic operators.

Complex types: lists, dicts, and JSON

The list and dict types are similar to their Python counterparts . FilterX uses JSON to represent generic dictionary and list types, but you can create other, specific dictionary and list types as well (currently for OTEL, for example, `otel_kvlist` , or `otel_array`). All supported dictionary and list types are compatible with each other, and you can convert them to and from each other, copy values between them (retaining the type), and so on.

For example:

```
my_list = []; # Creates an empty list (which defaults to a JSON list)
my_array = {}; # Creates an empty dictionary (which defaults to a JSON object)

my_list2 = json_array(); # Creates an empty JSON list
my_array2 = json(); # Creates an empty JSON object.
```

You can add elements to lists and dictionaries like this:

```
list = json_array(); # Create an empty JSON list
#list = otel_array(); # Create an OTEL list
list += ["first_element"]; # Append entries to the list
list += ["second_element"];
list += ["third_element"];
${MESSAGE} = list;
```

You can also create the list and assign values in a single step:

```
list = json_array(["first_element", "second_element", "third_element"]);
${MESSAGE} = list;
```

You can refer to the elements using an index (starting with `0`):

```
list = json_array(); # Create an empty JSON list
list[0] = "first_element"; # Append entries to the list
list[1] = "second_element";
list[2] = "third_element";
${MESSAGE} = list;
```

In all three cases, the value of `${MESSAGE}` is the same JSON array: `["first_element", "second_element", "third_element"]`.

You can define JSON objects using the `json()` type, for example:

```
js1 = json();
js1 += {
    "body": "mystring",
    "time_unix_nano": 123456789,
    "attributes": {
        "int": 42,
        "flag": true
        }
    };

js2 = json({"key": "value"})
```

Naturally, you can assign values from other variables to an object, for example:

```
js = json_array(["foo", "bar", "baz"]);
${MESSAGE} = json({
    "key": "value",
    "list": list
});
```

or

```
js = json({
    "key": ${MY-NAME-VALUE-PAIR},
    "key-from-expression": isset(${HOST}) ? ${HOST} : "default-hostname",
    "list": list
});
```

When working with dicts, note the following points:

- To create a field only if the assigned value is non-null, see Create dict element if non-null (`:??`) .
- To assign a value to a non-existing key where only this key doesn't exist, you can use a simple value assignment, for example:

```
js = json({
"key1": "one",
"key2": "two"
});

js.key3 = "three"
```

However, if you want to assign a value where multiple elements of the path don't exist, use the `dpath` FilterX function, for example:

```
dpath(js.key4.key41.key412) = "nested value"
```

The value of the dictionary will be:

```
js = json({
"key1": "one",
"key2": "two",
"key3": "three",
"key4": {
    "key41": {
        "key412": "nested value"
        }
   }
});
```

Within a FilterX block, you can access the fields of complex data types by using indexes and the dot notation, for example:

- dot notation: `js.key`
- indexing: `js["key"]`
- or mixed mode if needed: `js.list[1]`

When referring to the field of a name-value pair (which begins with the `$` character), place the dot or the square bracket outside the curly bracket surrounding the name of the name-value pair, for example: `${MY-LIST}[2]` or `${MY-OBJECT}.mykey`. If the name of the

key contains characters that are not permitted in FilterX variable names, for example, a hyphen (`-`), use the bracketed syntax and enclose the key in double quotes: `${MY-LIST}["my-key-name"]` .

You can add two lists or two dicts using the Plus operator.

Tuples

Available in AxoSyslog 4.27 and later.

A `tuple` is a read-only, list-like data type similar to a Python tuple. You can initialize a tuple only once, then it remains read-only until the end of its lifecycle.

Tuple examples:

```
t = (); # empty tuple
t = ("foo",); # singleton
t = (1,2,3); # a tuple of 3 elements

d = {'foo':'foovalue','bar':'barvalue'};
t = tuple([1,2,3,d]); # a tuple with the value (1,2,3,
{"foo":"foovalue","bar":"barvalue"})
```

List membership

Available in AxoSyslog 4.12 and later.

You can check whether a value is in a list using the `in` and `not in` membership operators. If you're checking the value of a variable that can have another type than the elements of a list, explicitly convert the value to the correct type (for example, string) before the comparison.

```
my_array = json_array(['hostname_one', 'hostname_two', 'hostname_three']);
if (${HOST} in my_array) {
    # ...
}
```

Starting with AxoSyslog 4.22, you can also use the operators to check membership in `dict` keys. For example:

```
my_dict = {"key-1": "value", "key-2": "another value"};
my_search_key = "key-1";
if (my_search_key in my_dict) {
    ${MSG} = "Found: " + my_dict[my_search_key];
} else {
    ${MSG} = "Not Found";
};
```

The `in` operator can be used to replicate the functionality of the in-list() filter function in FilterX. If you want to populate the list from a file, use the cache_json_file() FilterX function.

Type casting

To convert a value into another type, call the function named after the target type, for example, `int(${.apache.status})`. The following type casting functions are available:

- `bool`
- `bytes`
- `datetime`
- `dict` and `json`
- `double`
- `int`
- `list` and `json_array`
- `protobuf`
- `string`
- `tuple`

If a value can't be converted, the cast returns an error, and the statement is falsy. To use a default value instead, use the null coalescing operator , for example: `${.status} = int(${.apache.status}) ?? 0;`

FilterX variables in destinations

If you're modifying messages using FilterX (for example, you extract a value from the message and add it to another field of the message), note the following points:

- Macros and name-value pairs (variables with names beginning with the `$` character) are included in the outgoing message in case the template of the destination includes them. For example, if you change the value of the `${MESSAGE}` macro, it's automatically sent to the destination if the destination template includes this macro.
- Local and pipeline variables are not included in the message, you must assign their value to a macro or name-value pair that's included in the destination template to send them to the destination.
- When sending data to `opentelemetry()` destinations, if you're modifying messages received via the `opentelemetry()` source, then you must explicitly update the original (raw) data structures in your FilterX block, otherwise the changes won't be included in the outgoing message. For details, see Modify incoming OTEL.

10.3 Reuse FilterX blocks

Define a FilterX block once with block filterx, then call it by name from the filterx statements of multiple log paths.

To use a FilterX block in multiple log paths, you have to define it as a separate block:

```
block filterx <identifier>() {
    <filterx-statement-1>;
    <filterx-statement-2>;
    ...
};
```

Then use it in a log path:

```
log {
    source(s1);
    filterx { <identifier>(); };
    destination(d1);
};
```

For example, the following FilterX statement selects the messages that contain the word `deny` and come from the host `example`.

```
block filterx demo_filterx() {
    ${HOST} == "example";
    ${MESSAGE} =~ "deny";
};
log {
    source(s1);
    filterx { demo_filterx(); };
    destination(d1);
};
```

10.4 FilterX use cases and examples

Common FilterX tasks: set, delete, and rename message fields, conditional rewrites, and an iptables parser written in FilterX.

Common tasks

The following list shows you some common tasks that you can solve with FilterX:

- To set message fields (like macros or SDATA fields) or replace message parts: you can assign values to change parts of the message, or use one of the FilterX functions to rewrite existing values.

  > **Note**
  >
  > **Hard macros cannot be modified. For details, see Hard versus soft macros.**

- To delete or unset message fields, see Delete values .
- To rename a message field, assign the value of the old field to the new one, then unset the old field. For example:

  ```
  $my_new_field = $mike_old_field;
  unset($mike_old_field);
  ```

- To use conditional rewrites, you can either:
  - embed the FilterX block in an if-else block , or
  - use value comparison in the FilterX block to select the appropriate messages. For example, to rewrite only messages of the NGINX application, you can:

    ```
    ${PROGRAM} == "nginx";
    # <your rewrite expression>
    ```

Create an iptables parser

The following example shows you how to reimplement the iptables parser in a FilterX block. The following is a sample iptables log message (with line-breaks added for readability):

```
Dec 08 12:00:00 hostname.example kernel: custom-prefix:IN=eth0 OUT=
MAC=11:22:33:44:55:66:aa:bb:cc:dd:ee:ff:08:00 SRC=192.0.2.2 DST=192.168.0.1 LEN=40
TOS=0x00
PREC=0x00 TTL=232 ID=12345 PROTO=TCP SPT=54321 DPT=22 WINDOW=1023 RES=0x00 SYN
URGP=0
```

This is a normal RFC3164-formatted message logged by the kernel (where iptables logging messages originate from), and contains space-separated key-value pairs.

1. First, create some filter statements to select iptables messages only:

```
block filterx parse_iptables() {
    ${FACILITY} == "kern"; # Filter on the kernel facility
    ${PROGRAM} == "kernel"; # Sender application is the kernel
    ${MESSAGE} =~ "PROTO="; # The PROTO key appears in all iptables messages
}
```

2. To make the parsed data available under macros beginning with `${.iptables}`, like in the case of the original `iptables-parser()`, create the `${.iptables}` JSON object.

```
block filterx parse_iptables() {
    ${FACILITY} == "kern"; # Filter on the kernel facility
    ${PROGRAM} == "kernel"; # Sender application is the kernel
    ${MESSAGE} =~ "PROTO="; # The PROTO key appears in all iptables messages

    ${.iptables} = json(); # Create an empty JSON object
}
```

3. Add a key=value parser to parse the content of the messages into the `${.iptables}` JSON object. The key=value pairs are space-separated, while equal signs (=) separates the values from the keys.

```
block filterx parse_iptables() {
    ${FACILITY} == "kern"; # Filter on the kernel facility
    ${PROGRAM} == "kernel"; # Sender application is the kernel
    ${MESSAGE} =~ "PROTO="; # The PROTO key appears in all iptables messages

    ${.iptables} = json(); # Create an empty JSON object

    ${.iptables} = parse_kv(${MESSAGE}, value_separator="=", pair_separator="
");
}
```

For other examples on parsing messages, see the Parsing firewall logs with FilterX blog post.

10.5 Comparing values in FilterX

Compare macros, templates, and variables as strings or numbers in FilterX, including strict equality and inequality that also check the type.

In AxoSyslog you can compare macro values, templates, and variables as numerical and string values. String comparison is alphabetical: it determines if a string is alphabetically greater than or equal to another string. For details on macros and templates, see Customize message format using macros and templates.

Use the following syntax to compare macro values or templates.

```
filterx {
  "<macro-or-variable-or-expression>" operator "<macro-or-variable-or-expression>";
};
```

String and numerical comparison

You can use mathematical symbols as operators (like `==, !=, >=`), and based on the type of the arguments AxoSyslog automatically determines how to compare them. The logic behind this is similar to JavaScript:

- If both sides of the comparisons are strings, then the comparison is string.

  Note: Comparing strings is case sensitive. For case insensitive comparison, use the `strcasecmp()` function .

- If one of the arguments is numeric, then the comparison is numeric.
- Literal numbers (numbers not enclosed in quotes) are numeric.
- You can explicitly type-cast an argument into a number.
- The `bytes` , `json` , and `protobuf` types are always compared as strings.
- Currently you can't compare dictionaries and lists.

For example:

- `if (${.apache.httpversion} == 1.0)`

  The right side of the `==` operator is 1.0, which is a floating point literal (double), so the comparison is numeric.

- `if (double(${.apache.httpversion}) == "1.0")`

The left side is explicitly type cast into double, the right side is string (because of the quotes), so the comparison is numeric.

- `if (${.apache.request} == "/wp-admin/login.php")`

  The left side is not type-cast, the right side is a string, so the comparison is string, and case sensitive. The case insensitive equivalent is: `if (strcasecmp(${.apache.request}, "/wp-admin/login.php") == 0)`

> **Note**
>
> **You can use string operators if you want to, they are still available for backwards compatibility.**

Example: Compare macro values

The following expression selects log messages that contain a PID (that is, the `${PID}` macro is not empty):

```
filterx {
    ${PID};
};
```

(It is equivalent to using the `isset()` function: `isset(${PID});`).

The following expression selects log messages where the priority level is not `emerg` .

```
filterx {${LEVEL} != "emerg"; };
```

The following example selects messages with priority level higher than 5.

```
filterx {
    ${LEVEL_NUM} > 5;
};
```

Make sure to:

- Enclose literal strings and templates in double-quotes. For macros and variables do not use quotes.
- Use the `$` character before macros.

Note that you can use:

- type casting anywhere where you can use templates to apply a type to the result of the template expansion.
- any macro in the expression, including user-defined macros from parsers and classifications.
- boolean operators to combine comparison expressions.

Compare the type (strict equality)

To compare the values of operands and verify that they have the same type, use the `===` (strict equality) operator. The following example defines a string variable with the value “5” as string and uses it in different comparisons:

```
mystring = "5"; # Type is string
mystring === 5; # false, because the right-side is an integer
mystring === "5"; # true
};
```

To compare only the types of variables and macros, you can use the `istype` function .

Strict inequality operator

Compares the values of operands and returns `true` if they are different. Also returns `true` if the value of the operands are the same, but their type is different. For example:

```
"example" !== "example"; # False, because they are the same and both are strings
"1" !== 1; # True, because one is a string and the other an integer
```

Comparison operators

The following numerical and string comparison operators are available.

Numerical or string 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
===		Equals and has the same type
!==		Not equal to or has different type

10.6 Boolean operators in FilterX

Combine FilterX filter expressions with the and, or, and not operators, and invert conditions correctly using De Morgan's laws.

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

```
log {
    source(s1); source(s2);
    filterx { ${HOST} == "example1"; };
    filterx { ${HOST} == "example2"; };
    destination(d1); destination(d2); };
```

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

```
log {
    source(s1); source(s2);
    filterx { ${HOST} == "example1" or ${HOST} == "example2"; };
    destination(d1); destination(d2); };
```

Use the `not` operator to invert boolean filters, for example, to select messages that weren't sent by host `example1`:

```
filterx { not ( ${HOST} == "example1" ); };
```

> **Note**
>
> In some cases, instead of boolean operators, you can also use the `!=` (not equal to) comparison or the `!~` (doesn't contain) string operator.
>
> When checking for equality (`==`), sometimes it's also important to check that the two operands have the same type. For that purpose, you can use the `===` (strict equality) operator .

However, to select the messages that weren't sent by host `example1` or `example2`, you have to use the `and` operator (that's how boolean logic works, see De Morgan's laws for details):

```
filterx { not (${HOST} == "example1") and not (${HOST} == "example2"); };
```

Alternatively, you can use parentheses and the `or` operator to avoid this confusion:

```
filterx { not ( (${HOST} == "example1") or (${HOST} == "example2") ); };
```

The following filter statement selects the messages that contain the word `deny` and come from the host `example`.

```
filterx {
    ${HOST} == "example";
    ${MESSAGE} =~ "deny";
};
```

> **Note**
>
> **FilterX blocks are often used together with log path flags. For details, see Log path flags.**

10.7 Conditional statements in FilterX

Build decision trees in FilterX with if, elif, and else statements, or with switch-case expressions that also support integer ranges.

A conditional statement (`if (expr) { ... } elif (expr) {} else { ... };`) allows you to evaluate complex decision trees. For example:

```
if (${PRI} == "alert") {
  event.severity = 1;
} elif (${PRI} == "critical") {
  event.severity = 2;
} elif (${PRI} == "error") {
  event.severity = 3;
} elif (${PRI} == "warning") {
  event.severity = 4;
} elif (${PRI} == "notice") {
  event.severity = 5;
} elif (${PRI} == "information") {
  event.severity = 6;
} else {
  event.severity = 7;
};
```

Starting with version 4.10, you can also use switch-case expressions. Switch-case expressions allow you to better organize the code instead of using multiple `if` , `elif` , `else` blocks. Using switch-case expressions also improves performance.

Usage:

```
  switch (<expression-or-variable>) {
    case "literal-string1":
      ...
      break;
    case "literal-string2":
      ...
      break;
    case <expression>:
      ...
      break;
    default:
      ...
      break;
};
```

For example:

```
  switch ($MESSAGE) {
    case "foobar":
      $MESSAGE = "literal-case";
      break;
    case any_expression:
      $MESSAGE = "variable-case";
      break;
    default:
      $MESSAGE = "default";
      break;
};
```

For integer targets, in version 4.26 and later you can use ranges, for example:

```
switch (${values.int}) {
    case 1..4:
        result = "within range";
        break;
    case 5:
        result = "exact value";
        break;
    default:
        result = "default";
        break;
};
```

Note that:

- Cases with literal string targets are stored in a map, and the lookup is started with them.
- Case targets can contain any expressions, and they are evaluated in order.
- Literal string and default target duplications are checked and will cause init failure. Non-literal expression targets are not checked, and only the first matching case will be executed.

10.8 String search in FilterX

Search strings with the includes(), startswith(), and endswith() FilterX functions, and replace substrings with str_replace() or regexp_subst().

Available in AxoSyslog 4.9 and later.

You can check if a string contains a specified string using the `includes` FilterX function. The `startswith` and `endswith` functions check the beginning and ending of the strings, respectively. For example, the following expression checks if the message (`$MESSAGE`) begins with the `%ASA-` string:

```
startswith($MESSAGE, '%ASA-')
```

By default, matches are case sensitive. For case insensitive matches, use the `ignorecase=true` option:

```
startswith($MESSAGE, '%ASA-', ignorecase=true)
```

All three functions (`includes` , `startswith` , and `endswith`) can take a list with multiple search strings and return true if any of them match. This is equivalent with using combining the individual searches with logical OR operators. For example:

```
${MESSAGE} = "%ASA-5-111010: User ''john'', running ''CLI'' from IP 0.0.0.0,
executed ''dir disk0:/dap.xml"
includes($MESSAGE, ['%ASA-','john','CLI'])

includes($MESSAGE, ['%ASA-','john','CLI'])
includes($MESSAGE, '%ASA-') or includes($MESSAGE, 'john') or includes($MESSAGE,
'CLI')
```

Starting with AxoSyslog version 4.19, the `includes` function has a `limit` option to truncate the search to the first `<limit>` character of the string. The following example searches for the string `john` only in the first 40 characters of the `$MESSAGE` :

```
${MESSAGE} = "%ASA-5-111010: User ''john'', running ''CLI'' from IP 0.0.0.0,
executed ''dir disk0:/dap.xml"
includes($MESSAGE, 'john', limit=40)
```

For more complex searches, or if you need to match a regular expression, use the `regexp_search` FilterX function .

String replace

FilterX provides two functions to replace parts of a string: `str_replace` for simple string search, and `regexp_subst` for more complex replacements.

10.9 IP addresses and subnets

Represent IPv4 and IPv6 addresses and subnets with the ip() and subnet() FilterX types, and filter messages by source network.

Available in AxoSyslog 4.25 and later.

FilterX has two types to work with network addresses:

- `ip()` represents a single IPv4 or IPv6 address, while
- `subnet()` represents an IPv4 or IPv6 subnet in CIDR notation.

You can use these types together with the list membership operator (`in`) to check whether an address belongs to a subnet, which is useful for filtering messages based on the source or destination network of the traffic they describe.

ip

The `ip()` type represents a single IPv4 or IPv6 address as a string. AxoSyslog returns an error if the argument cannot be parsed as a valid IP address.

For example:

```
filterx {
    addr4 = ip("192.168.2.10");
    addr6 = ip("2001:db8::1");
};
```

You can also pass a variable that contains the address as a string:

```
filterx {
    src = "192.168.2.10";
    addr = ip(src);
};
```

subnet

The `subnet()` type represents an IPv4 or IPv6 subnet as a string in CIDR notation. AxoSyslog returns an error if the argument cannot be parsed as a valid subnet.

- For IPv4, you can use either prefix length notation (for example, `/24`) or a full netmask (for example, `/255.255.255.0`).
- For IPv6, only prefix length notation is accepted (for example, `/64`).

If the address part contains host bits, AxoSyslog masks them out so the resulting subnet always represents the network address. For example, `subnet("192.0.2.5/24")` is equivalent to `subnet("192.0.2.0/24")`.

For example:

```
filterx {
    net4 = subnet("192.0.2.0/24");
    net4_netmask = subnet("192.0.2.0/255.255.255.0");
    net6 = subnet("2001:db8::/32");
};
```

Check whether an IP address is in a subnet

To check whether an IP address belongs to a subnet, use the list membership operator (`in`). The left-hand side can be a string that contains an IP address, or an `ip()` object. The right-hand side must be a `subnet()` object.

The following example checks whether a string IP address is in an IPv4 subnet:

```
filterx {
    net = subnet("192.0.2.0/24");
    result = json();
    result.member = "192.0.2.100" in net;     # true
    result.non_member = "198.51.100.1" in net; # false
    ${MESSAGE} = result;
};
```

The same works for IPv6 subnets:

```
filterx {
    net = subnet("2001:db8::/32");
    result = json();
    result.member = "2001:db8::1" in net;     # true
    result.non_member = "2001:db9::1" in net; # false
    ${MESSAGE} = result;
};
```

You can also use an `ip()` object on the left-hand side, which is useful when the address is already stored as an `ip()` value:

```
filterx {
    net = subnet("192.0.2.0/24");
    addr = ip("192.0.2.100");
    addr in net; # true
};
```

CAUTION:

The address family of the IP and the subnet must match. Comparing an IPv4 address to an IPv6 subnet (or the other way around) evaluates to false, so the surrounding statement is treated as falsy and the message is dropped from the log path. Wrap the check in a conditional if you want to keep processing the message in this case.

Filter messages based on the source network

A typical use case is to filter or tag messages based on the network the message originated from. The following example tags messages that come from the `192.168.2.0/24` internal network:

```
filterx {
    internal_net = subnet("192.168.2.0/24");
    ${labels} = json();
    if (${SOURCEIP} in internal_net) {
        ${labels.network} = "internal";
    } else {
        ${labels.network} = "external";
    };
};
```

You can combine multiple subnets with logical operators to match a list of allowed networks:

```
filterx {
    office = subnet("192.0.2.0/24");
    datacenter = subnet("198.51.100.0/24");
    ${labels} = json();
    if (${SOURCEIP} in office or ${SOURCEIP} in datacenter) {
        ${labels.network} = "trusted";
    };
};
```

10.10 Parsing data in FilterX

Parse CEF, CSV, date, JSON, key=value, LEEF, XML, and Windows Event Log XML data into FilterX dicts, lists, and datetime values.

10.10.1 Parse CEF with FilterX

Parse Common Event Format (CEF) messages into a dict with the parse_cef() FilterX function, and set its pair and value separators.

Available in AxoSyslog 4.9 and later.

The `parse_cef` FilterX function parses messages formatted in the Common Event Format (CEF) into a JSON object.

To create CEF messages from FilterX data, see Common Event Format (CEF).

Declaration

Usage: `parse_cef(<input-string>, value_separator="=", pair_separator="|", separate_extensions=false)`

The first argument is the input message. Optionally, you can set the `pair_separator` and `value_separator` arguments to override their default values.

The `value_separator` must be a single-character string. The `pair_separator` can be a regular string.

Starting with version 4.13, AxoSyslog parses fields from extensions to the same level as regular fields. In earlier versions, extensions were grouped under the `extensions` key. To keep using the `extensions` key, set `separate_extensions=true` .

The parsed JSON object has the following fields:

- `cef_version`
- `device_vendor`
- `device_product`
- `device_version`
- `device_event_class_id`
- `event_name`
- `agent_severity`
- `extensions`

> **Note**
>
> The name of some fields changed in the parsed object in version 4.16 for clarity, and to avoid name collisions with fields in the extensions:
>
> - `version` -> `cef_version`
> - `name` -> `event_name`

Example

The following is a CEF-formatted message including mandatory and custom (extension) fields:

```
CEF:0|KasperskyLab|SecurityCenter|13.2.0.1511|KLPRCI_TaskState|Completed
successfully|1|foo=foo bar=bar baz=test
```

The following FilterX expression parses it, and stores the result in the `${PARSED_MESSAGE}` name-value pair. Assigning a dict to a name-value pair stores it as JSON, so you don't have to format it explicitly:

```
filterx {
    ${PARSED_MESSAGE} = parse_cef(${MESSAGE});
};
```

The content of the JSON object for this message will be:

```
{
"cef_version":"0",
"device_vendor":"KasperskyLab",
"device_product":"SecurityCenter",
"device_version":"13.2.0.1511",
"device_event_class_id":"KLPRCI_TaskState",
"name":"Completed successfully",
"agent_severity":"1",
"foo":"foo=bar",
"bar":"bar=baz",
"baz":"test"
}
```

If you set `separate_extensions=true`, the extensions of the message will be grouped under the `extensions` key:

```
{
"cef_version":"0",
"device_vendor":"KasperskyLab",
"device_product":"SecurityCenter",
"device_version":"13.2.0.1511",
"device_event_class_id":"KLPRCI_TaskState",
"name":"Completed successfully",
"agent_severity":"1",
"extensions": {
    "foo":"foo=bar",
    "bar":"bar=baz",
    "baz":"test"
    }
}
```

Options of CEF parsers

The `parse_cef` FilterX function has the following options.

pair_separator

Specifies the character or string that separates the key-value pairs in the extensions. Default value: ` ` (space).

separate_extensions

Available in AxoSyslog 4.13 and later.

Starting with version 4.13, AxoSyslog parses fields from extensions to the same level as regular fields. In earlier versions, extensions were grouped under the `extensions` key. To keep using the `extensions` key, set `separate_extensions=true`.

Default value: `false`

value_separator

Specifies the character that separates the keys from the values in the extensions. Default value: `=`.

10.10.2 Parse comma-separated values with FilterX

Split messages along delimiters into lists or dicts with the parse_csv() FilterX function, with examples for Apache logs and message segments.

The `parse_csv` FilterX function can separate parts of log messages (that is, the contents of the `${MESSAGE}` macro) along delimiter characters or strings into lists, or key-value pairs within dictionaries, using the csv (comma-separated-values) parser.

Usage: `parse_csv(<input-string>, columns=json_array, delimiter=string, string_delimiters=json_array, dialect=string, strip_whitespace=boolean, greedy=boolean)`

Only the input parameter is mandatory.

If the `columns` option is set, `parse_csv` returns a dictionary with the column names (as keys) and the parsed values. If the `columns` option isn't set, `parse_csv` returns a list.

To create comma-separated values from FilterX data, see Format data as comma-separated values.

The following example separates hostnames like `example-1` and `example-2` into two parts.

```
block filterx p_hostname_segmentation() {
    cols = json_array(["NAME","ID"]);
    HOSTNAME = parse_csv(${HOST}, delimiter="-", columns=cols);
    # HOSTNAME is a json object containing parts of the hostname
    # For example, for example-1 it contains:
    # {"NAME":"example","ID":"1"}

    # Set the important elements as name-value pairs so they can be referenced in
the destination template
    ${HOSTNAME_NAME} = HOSTNAME.NAME;
    ${HOSTNAME_ID} = HOSTNAME.ID;
};
destination d_file {
    file("/var/log/${HOSTNAME_NAME:-examplehost}/${HOSTNAME_ID}"/messages.log);
};
log {
    source(s_local);
    filterx { p_hostname_segmentation(); };
    destination(d_file);
};
```

Parse 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 (`delimiter=" "`). Excess leading and trailing whitespace characters are stripped.

```
block filterx p_apache() {
    ${APACHE} = json();
    cols = [
    "CLIENT_IP", "IDENT_NAME", "USER_NAME",
    "TIMESTAMP", "REQUEST_URL", "REQUEST_STATUS",
    "CONTENT_LENGTH", "REFERER", "USER_AGENT",
    "PROCESS_TIME", "SERVER_NAME"
    ];
    ${APACHE} = parse_csv(${MESSAGE}, columns=cols, delimiter=(" "),
strip_whitespace=true, dialect="escape-double-char");

    # Set the important elements as name-value pairs so they can be referenced in
the destination template
    ${APACHE_USER_NAME} = ${APACHE.USER_NAME};
};
```

The results can be used for example, to separate log messages into different files based on the APACHE.USER_NAME field. in case the field is empty, the `nouser` string is assigned as default.

```
log {
    source(s_local);
    filterx { p_apache(); };
    destination(d_file);
};
destination d_file {
    file("/var/log/messages-${APACHE_USER_NAME:-nouser}");
};
```

Segment a part of a message

You can use multiple parsers in a layered manner to split parts of an already parsed message into further segments. The following example splits the timestamp of a parsed Apache log message into separate fields. Note that the scoping of FilterX variables is important:

- If you add the new parser to the FilterX block used in the previous example , every variable is available.
- If you use a separate FilterX block, only global variables and name-value pairs (variables with names starting with the `$` character) are accessible from the block.

```
block filterx p_apache_timestamp() {
    cols = ["TIMESTAMP.DAY", "TIMESTAMP.MONTH", "TIMESTAMP.YEAR", "TIMESTAMP.HOUR",
"TIMESTAMP.MIN", "TIMESTAMP.SEC", "TIMESTAMP.ZONE"];
    ${APACHE.TIMESTAMP} = parse_csv(${APACHE.TIMESTAMP}, columns=cols,
delimiter="/: ", dialect="escape-none");

    # Set the important elements as name-value pairs so they can be referenced in
the destination template
    ${APACHE_TIMESTAMP_DAY} = ${APACHE.TIMESTAMP_DAY};
};
destination d_file {
    file("/var/log/messages-${APACHE_USER_NAME:-nouser}/${APACHE_TIMESTAMP_DAY}");
};
log {
    source(s_local);
    filterx { p_apache(); };
    filterx { p_apache_timestamp(); };
    destination(d_file);
};
```

10.10.2.1 Options of the parse_csv FilterX function

Reference of the parse_csv() FilterX function options: columns, delimiter, dialect, greedy, quote_pairs, strip_whitespace, and string_delimiters.

The `parse_csv` FilterX function has the following options.

columns

Synopsis:	`columns=["1st","2nd","3rd"]`
Default value:	N/A

Description: Specifies the names of the columns, and correspondingly the keys in the resulting JSON array.

- If the `columns` option is set, `parse_csv` returns a dictionary with the column names (as keys) and the parsed values.
- If the `columns` option isn't set, `parse_csv` returns a list.

delimiter

Synopsis:	`delimiter="<string-with-delimiter-characters>"`
Default value:	`,`

Description: The delimiter parameter contains the characters that separate the columns in the input string. If you specify multiple characters, every character will be treated as a delimiter. Note that the delimiters aren't included in the column values. For example:

- To separate the text at every hyphen (-) and colon (:) character, use `delimiter="-:"`.
- To separate the columns along the tabulator (tab character), specify `delimiter="\\t"`.
- To use strings instead of characters as delimiters, see `string_delimiters`.

Multiple delimiters

If you use more than one delimiter, note the following points:

- AxoSyslog 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 may 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 input, AxoSyslog uses the string delimiter.

dialect

Synopsis:	`dialect="<dialect-name>"`
Default value:	`escape-none`

Description: Specifies how to handle escaping in the input strings.

The following values are available.

- `escape-backslash` : The parsed message uses the backslash (`\\`) character to escape quote characters.
- `escape-backslash-with-sequences` : The parsed message uses `""` as an escape character but also supports C-style escape sequences, like `\n` or `\r` . Available in AxoSyslog version 4.0 and later.
- `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

Synopsis:	`greedy=true`
Default value:	`false`

If the `greedy` option is enabled, AxoSyslog adds the remaining part of the message to the last column, ignoring any delimiters that may appear in this part of the message. You can use this option to process messages where the number of columns varies from message to message.

For example, you receive the following comma-separated message: `example 1, example2, example3`, and you segment it with the following parser:

```
my-parsed-values = parse_csv(${MESSAGE}, columns=["COLUMN1", "COLUMN2", "COLUMN3"],
delimiter=",");
```

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 thus possibly lost if you use only the parsed columns to reconstruct the message (for example, if you send the columns to different columns of a database table).

Using the `greedy=true` 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`.

```
my-parsed-values = parse_csv(${MESSAGE}, columns=["COLUMN1", "COLUMN2", "COLUMN3"],
delimiter=",", greedy=true);
```

quote_pairs

Synopsis: `quote_pairs=["<quote-pair1>", "<quote-pair2>"]`

Available in AxoSyslog 4.18 and later.

Description: List of quote pairs that are ignored and removed from the beginning and end of the strings. Note that the beginning and ending quote character does not have to be identical, for example, `[}` can also be a quote-pair.

In the following example, square brackets (`[]`) and single-quotes (`'`) are ignored:

```
filterx {
  str = "value1,[value2],'value3'";
  ${MESSAGE} = parse_csv(str, quote_pairs=["[]", "'"]);
  # The value of ${MESSAGE} will be "value1,value2,value3"
};
```

strip_whitespace

Synopsis:	`strip_whitespace=true`
Default value:	`false`

Description: Remove leading and trailing whitespaces from all columns. The `strip_whitespaces` option is an alias for `strip_whitespace`.

string_delimiters

Synopsis:	`string_delimiters=json_array(["first-string","2nd-string"])`

Description: In case you have to use a string as a delimiter, list your string delimiters as a JSON array in the `string_delimiters=["<delimiter_string1>", "<delimiter_string2>", ...]` option.

By default, the `parse_csv` FilterX function uses the comma as a delimiter. If you want to use only strings as delimiters, you have to disable the default comma delimiter, for example: `delimiter="", string_delimiters=["<delimiter_string>"]`

Otherwise, AxoSyslog will use the string delimiters in addition to the default character delimiter, so for example, `string_delimiters=["=="]` is actually equivalent to `delimiter=",", string_delimiters=["=="]`, and not `delimiter="", string_delimiters=["=="]`

Multiple delimiters

If you use more than one delimiter, note the following points:

- AxoSyslog 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 may 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 input, AxoSyslog uses the string delimiter.

10.10.3 Parse dates and timestamps with FilterX

Parse date strings into datetime values with the strptime() FilterX function, and set the timestamp of the message with set_timestamp().

To parse a date or a timestamp in FilterX, use the `strptime` function. It creates a `datetime` value from a string, using one or more format strings. It's the FilterX equivalent of the `date-parser()` .

To use the parsed value as the timestamp of the message, pass it to the `set_timestamp` function.

For example, if the message contains key=value pairs like `ts=2024-04-10T08:09:10+0200 user=alice` , the following FilterX block parses the `ts` field, and sets it as the timestamp of the message:

```
filterx {
    fields = parse_kv(${MESSAGE}, value_separator="=", pair_separator=" ");
    ts = strptime(fields.ts, "%Y-%m-%dT%H:%M:%S%z", "%d/%b/%Y:%H:%M:%S%z");
    isset(ts);
    set_timestamp(ts);
};
```

- `strptime` tries the format strings in order. In this example, it accepts both `2024-04-10T08:09:10+0200` and `10/Apr/2024:08:09:10+0200` . For the list of format codes, see `strptime` .
- If none of the formats match, `strptime` returns null. The `isset` statement then makes the FilterX block false, so AxoSyslog doesn't send messages with an unparsable date.
- By default, `set_timestamp` sets the timestamp of the message (`stamp`). To set the time when AxoSyslog received the message instead, use `set_timestamp(ts, stamp="recvd")` .

After the block, the date macros of the message, like `${ISODATE}` , use the parsed timestamp.

If the timezone of your dates is missing or wrong, see Handle and fix timezones and timestamps.

To format a `datetime` value as a string, use the `strftime` or the `format_isodate` function.

> **Note**
>
> **To replace an existing `date-parser()` with FilterX, see Update to FilterX.**

10.10.4 Parse JSON with FilterX

Parse JSON strings into FilterX objects with the parse_json() function, then access their fields.

To parse a JSON string, use the `parse_json` function. It returns a value of the matching FilterX type: a JSON object becomes a dict, an array becomes a list. To convert an object back to a JSON string, use the `format_json` function.

If the string isn't valid JSON, the function returns an error, and the FilterX block evaluates to false.

For example, if the message contains a JSON object like `{"user":"alice","action":"login","src":{"ip":"192.0.2.2"}}`, the following FilterX block parses it and stores two of its fields in name-value pairs:

```
filterx {
    event = parse_json(${MESSAGE});
    ${USER} = event.user;
    ${SRC_IP} = event.src.ip;
};
```

After the block, `${USER}` is `alice`, and `${SRC_IP}` is `192.0.2.2`. The `event` variable is a local variable, so it isn't sent to the destination. For details, see FilterX variables in destinations .

To access the fields of the parsed object, see Complex types: lists, dicts, and JSON .

> **Note**
>
> **To replace an existing `json-parser()` with FilterX, see Update to FilterX.**

10.10.5 Parse key=value pairs with FilterX

Split key=value pairs, like Postfix or MySQL logs, into a dict with the parse_kv() FilterX function, and handle repeated keys and stray words.

The `parse_kv` FilterX function can split a string consisting of whitespace or comma-separated `key=value` pairs (for example, Postfix log messages). You can also specify other value separator characters instead of the equal sign, for example, colon (`:`) to parse MySQL log messages. The AxoSyslog application automatically trims any leading or trailing whitespace characters from the keys and values, and also parses values that contain unquoted whitespace.

To create key=value pairs from FilterX data, see Key-value pairs.

> **Note**
>
> **If a log message contains the same key multiple times (for example, `key1=value1, key2=value2, key1=value3, key3=value4, key1=value5`), then AxoSyslog only stores the last (rightmost) value for the key. Using the previous example, AxoSyslog will store the following pairs: `key1=value5, key2=value2, key3=value4` .**

> **Warning**
>
> By default, the parser discards sections of the input string that are not `key=value` pairs, even if they appear between `key=value` pairs that can be parsed. To store such sections, see stray_words_key.
>
> The names of the keys can contain only the following characters: numbers (0-9), letters (a-z,A-Z), underscore (_), dot (.), hyphen (-). Other special characters are not permitted.

Declaration

Usage: `parse_kv(<input-string>, value_separator="=", pair_separator=",", stray_words_key="stray_words")`

The `value_separator` must be a single-character string. The `pair_separator` can be a regular string.

Example

In the following example, the source is a Postfix log message consisting of comma-separated `key=value` pairs:

```
Jun 20 12:05:12 mail.example.com <info> postfix/qmgr[35789]: EC2AC1947DA: from=
<me@example.com>, size=807, nrcpt=1 (queue active)
```

```
filterx {
    ${PARSED_MESSAGE} = parse_kv(${MESSAGE});
};
```

You can set the value separator character (the 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;'
```

```
filterx {
    ${PARSED_MESSAGE} = parse_kv(${MESSAGE}, value_separator=":",
pair_separator=",");
};
```

10.10.5.1 Options of the parse_kv FilterX function

Reference of the parse_kv() FilterX function options: pair_separator, value_separator, stray_words_key, and stray_words_append_to_value.

The `parse_kv` FilterX function has the following options.

pair_separator

Specifies the character or string that separates the key-value pairs from each other. Default value: `,` .

For example, to parse `key1=value1;key2=value2` pairs, use:

```
${MESSAGE} = parse_kv("key1=value1;key2=value2", pair_separator=";");
```

stray_words_append_to_value

Available in AxoSyslog 4.17 and later.

If the `stray_words_append_to_value` flag is set, any stray words between the value pairs are appended to the preceding value. For example:

```
# input: a=b b=c d e f=g
filterx {
  ${MESSAGE} = parse_kv(${MESSAGE}, value_separator="=", pair_separator=" ",
stray_words_append_to_value=true);
};
# The value of $MSG will be: {"a":"b","b":"c d e","f":"g"}
```

If you want to collect the stray words into a separate key, see `stray_words_key` .

> **Note**
>
> **Note that you cannot use `stray_words_append_to_value` and `stray_words_key` in the same parser.**

stray_words_key

Specifies the key where AxoSyslog stores any stray words that appear before or between the parsed key-value pairs. If multiple stray words appear in a message, then AxoSyslog stores them as a comma-separated list. Default value: `N/A`

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;` and `inbound;`. If you want to store or process these, specify a key to store them, for example:

```
${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;";
${PARSED_MESSAGE} = parse_kv(${MESSAGE}, stray_words_key="stray_words");
```

The value of `${PARSED_MESSAGE}.stray_words` for this message will be: `["interzone-emtn_s1_vpn-enodeb_om", "inbound"]`

If you want to append the stray words to the respective values instead of adding them to a separate value, see `stray_words_append_to_value` .

> **Note**
>
> **Note that you cannot use `stray_words_append_to_value` and `stray_words_key` in the same parser.**

value_separator

Specifies the character that separates the keys from the values. Default value: `=` .

For example, to parse `key:value` pairs, use:

```
${MESSAGE} = parse_kv("key1:value1,key2:value2", value_separator=":");
```

10.10.6 Parse LEEF with FilterX

Parse LEEF 1.0 and 2.0 messages into a dict with the parse_leef() FilterX function, and set its pair and value separators.

Available in AxoSyslog 4.9 and later.

The `parse_leef` FilterX function parses messages formatted in the Log Event Extended Format (LEEF) into a JSON object.

Both LEEF versions (1.0 and 2.0) are supported.

To create LEEF messages from FilterX data, see Log Event Extended Format (LEEF).

Declaration

Usage: `parse_leef(<input-string>, value_separator="=", pair_separator="|", separate_extensions=false)`

The first argument is the input message. Optionally, you can set the `pair_separator` and `value_separator` arguments to override their default values.

The `value_separator` must be a single-character string. The `pair_separator` can be a regular string.

The parsed JSON object has the following fields:

- For LEEF v1 messages:
  - `leef_version`
  - `vendor_name`
  - `product_name`
  - `product_version`
  - `event_id`
  - `extensions`
- For LEEF v2 messages:
  - `leef_version`
  - `vendor_name`
  - `product_name`
  - `product_version`
  - `event_id`
  - `leef_delimiter`

- `extensions`

> **Note**
>
> The name of some fields changed in the parsed object in version 4.16 for clarity, and to avoid name collisions with fields in the extensions:
>
> - `version` -> `leef_version`
> - `vendor` -> `vendor_name`
> - `delimiter` -> `leef_delimiter`

Example

The following is a LEEF-formatted message including mandatory and custom (extension) fields:

```
LEEF:1.0|Microsoft|MSExchange|4.0 SP1|15345|src=192.0.2.0       dst=172.50.123.1
sev=5cat=anomaly        srcPort=81      dstPort=21      usrName=john.smith
```

The following FilterX expression parses it, and stores the result in the `${PARSED_MESSAGE}` name-value pair. Assigning a dict to a name-value pair stores it as JSON, so you don't have to format it explicitly:

```
filterx {
    ${PARSED_MESSAGE} = parse_leef(${MESSAGE});
};
```

The content of the JSON object for this message will be:

```
{
"leef_version":"1.0",
"vendor_name":"Microsoft",
"product_name":"MSExchange",
"product_version":"4.0 SP1",
"event_id":"15345",
"src":"192.0.2.0",
"dst":"172.50.123.1",
"sev":"5cat=anomaly",
"srcPort":"81",
"dstPort":"21",
"usrName":"john.smith"
}
```

If you set `separate_extensions=true`, the extensions of the message will be grouped under the `extensions` key:

```
{
"leef_version":"1.0",
"vendor_name":"Microsoft",
"product_name":"MSExchange",
"product_version":"4.0 SP1",
"event_id":"15345",
"extensions": {
    "src":"192.0.2.0",
    "dst":"172.50.123.1",
    "sev":"5cat=anomaly",
    "srcPort":"81",
    "dstPort":"21",
    "usrName":"john.smith"
    }
}
```

Options of LEEF parsers

The `parse_leef` FilterX function has the following options.

pair_separator

Specifies the character or string that separates the key-value pairs in the extensions. Default value: `\t` (tab).

LEEF v2 can specify the separator per message. Omitting this option uses the LEEF v2 provided separator, setting this value overrides it during parsing.

separate_extensions

Available in AxoSyslog 4.13 and later.

Starting with version 4.13, AxoSyslog parses fields from extensions to the same level as regular fields. In earlier versions, extensions were grouped under the `extensions` key. To keep using the `extensions` key, set `separate_extensions=true`.

Default value: `false`

value_separator

Specifies the character that separates the keys from the values in the extensions. Default value: `=`.

10.10.7 Parse Windows Event Log XML with FilterX

Parse Windows Event Log XML into a dict with the parse_windows_eventlog_xml() FilterX function, and see when the parser fails.

Available in AxoSyslog 4.9 and later.

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

The `parse_windows_eventlog_xml()` FilterX function parses Windows Event Logs XMLs. It's a specialized version of the `parse_xml()` parser .

To create Windows Event Log XML from FilterX data, see Windows Event Logs XML.

The parser returns false in the following cases:

- The input isn't valid XML.
- The root element doesn't reference the Windows Event Log schema (`<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'>`). Note that the parser doesn't validate the input data to the schema.

For example, the following converts the input XML into a JSON object:

```
filterx {
  xml = "<xml-input/>"
  $MSG = parse_windows_eventlog_xml(xml);
};
```

Given the following input:

```
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'>
    <System>
        <Provider Name='EventCreate'/>
        <EventID Qualifiers='0'>999</EventID>
        <Version>0</Version>
        <Level>2</Level>
        <Task>0</Task>
        <Opcode>0</Opcode>
        <Keywords>0x80000000000000</Keywords>
        <TimeCreated SystemTime='2024-01-12T09:30:12.1566754Z'/>
        <EventRecordID>934</EventRecordID>
        <Correlation/>
        <Execution ProcessID='0' ThreadID='0'/>
        <Channel>Application</Channel>
        <Computer>DESKTOP-2MBFIV7</Computer>
        <Security UserID='S-1-5-21-3714454296-2738353472-899133108-1001'/>
    </System>
    <RenderingInfo Culture='en-US'>
        <Message>foobar</Message>
        <Level>Error</Level>
        <Task></Task>
        <Opcode>Info</Opcode>
        <Channel></Channel>
        <Provider></Provider>
        <Keywords>
            <Keyword>Classic</Keyword>
        </Keywords>
    </RenderingInfo>
    <EventData>
        <Data Name='param1'>foo</Data>
        <Data Name='param2'>bar</Data>
    </EventData>
</Event>
```

The parser creates the following JSON object:

```
{
    "Event": {
        "@xmlns": "http://schemas.microsoft.com/win/2004/08/events/event",
        "System": {
            "Provider": {"@Name": "EventCreate"},
            "EventID": {"@Qualifiers": "0", "#text": "999"},
            "Version": "0",
            "Level": "2",
            "Task": "0",
            "Opcode": "0",
            "Keywords": "0x80000000000000",
            "TimeCreated": {"@SystemTime": "2024-01-12T09:30:12.1566754Z"},
            "EventRecordID": "934",
            "Correlation": "",
            "Execution": {"@ProcessID": "0", "@ThreadID": "0"},
            "Channel": "Application",
            "Computer": "DESKTOP-2MBFIV7",
            "Security": {"@UserID": "S-1-5-21-3714454296-2738353472-899133108-
1001"},
        },
        "RenderingInfo": {
            "@Culture": "en-US",
            "Message": "foobar",
            "Level": "Error",
            "Task": "",
            "Opcode": "Info",
            "Channel": "",
            "Provider": "",
            "Keywords": {"Keyword": "Classic"},
        },
        "EventData": {
            "Data": {
                "param1": "foo",
                "param2": "bar",
            },
        },
    },
}
```

10.10.8 Parse XML with FilterX

Parse raw XML into a dict with the parse_xml() FilterX function, and see how elements, attributes, and text values are mapped to dict keys.

Available in AxoSyslog 4.9 and later.

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

The `parse_xml()` FilterX function parses raw XMLs into dictionaries. This is a new implementation, so the limitations and options of the legacy `xml-parser()` do not apply.

To create XML from FilterX data, see Format data as XML.

There is no standardized way of converting XML into a dict. AxoSyslog creates the most compact dict possible. This means certain nodes will have different types and structures depending on the input XML element. Note the following points:

1. Empty XML elements become empty strings.

```
XML:  <foo></foo>
JSON: {"foo": ""}
```

2. Attributions are stored in `@attr` key-value pairs, similarly to other converters (like python xmltodict).

```
XML:  <foo bar="123" baz="bad"/>
JSON: {"foo": {"@bar": "123", "@baz": "bad"}}
```

3. If an XML element has both attributes and a value, we need to store them in a dict, and the value needs a key. We store the text value under the `#text` key.

```
XML:  <foo bar="123">baz</foo>
JSON: {"foo": {"@bar": "123", "#text": "baz"}}
```

4. An XML element can have both a value and inner elements. We use the `#text` key here, too.

```
XML:  <foo>bar<baz>123</baz></foo>
JSON: {"foo": {"#text": "bar", "baz": "123"}}
```

5. An XML element can have multiple values separated by inner elements. In that case we concatenate the values.

```
XML:  <foo>bar<a></a>baz</foo>
JSON: {"foo": {"#text": "barbaz", "a": ""}}
```

Usage

```
my_structured_data = parse_xml(raw_xml);
```

10.11 Format data

Format FilterX data as CEF, CSV, JSON, key=value pairs, LEEF, RFC5424 syslog, XML, or Windows Event Log XML strings.

AxoSyslog FilterX has several functions to format data into specific formats.

10.11.1 Common Event Format (CEF)

Format a FilterX dictionary into a Common Event Format (CEF) message with the format_cef() function.

Available in AxoSyslog 4.13 and later.

Formats a dictionary into the Common Event Format (CEF) .

To parse CEF messages, see Parse CEF with FilterX.

Usage: `${MESSAGE} = format_cef(my_dictionary);`

For example:

```
my_dictionary = {"version":"0","device_vendor":" KasperskyLab
","device_product":"SecurityCenter","device_version":"13.2.0.1511","device_event_cl
ass_id":"KLPRCI_TaskState","name":"Completed successfully","agent_severity":"1"};
```

Becomes:

```
CEF:0| KasperskyLab |SecurityCenter|13.2.0.1511|KLPRCI_TaskState|Completed
successfully|1|
```

The following keys must be available in the dictionary, otherwise formatting fails with an error message like: `FILTERX ERROR; ....| format_cef(my_dictionary)', error='Failed to evaluate event formatter function:` .

`cef_version` , `device_vendor` , `device_product` , `device_version` , `device_event_class_id` , `event_name` , `agent_severity` , `extensions`

10.11.2 Format data as comma-separated values

Format a FilterX dict or list into a delimiter-separated string with format_csv(), and set the columns, delimiter, and default value.

Formats a dictionary or a list into a comma-separated string.

To parse comma-separated values, see Parse comma-separated values with FilterX.

Usage: `format_csv(<input-list-or-dict>, columns=<json-list>, delimiter=<delimiter-character>, default_value=<string>)`

Only the input is mandatory, other arguments are optional. Note that the delimiter must be a single character.

By default, the delimiter is the comma (`delimiter=","`), the `columns` and `default_value` are empty.

If the `columns` option is set, AxoSyslog checks that the number of fields or entries in the input data matches the number of columns. If there are fewer items, it adds the `default_value` to the missing entries.

10.11.3 JSON

Serialize any FilterX value, including dicts and lists, into a JSON string with the format_json() function.

Formats any value into a raw JSON string.

To parse JSON strings, see Parse JSON with FilterX.

Usage: `format_json($data)`

10.11.4 Key-value pairs

Format a FilterX dictionary as key=value pairs with format_kv(), with custom value and pair separators and quoting of values.

Formats a dictionary into a string containing key=value pairs.

To parse key=value pairs, see Parse key=value pairs with FilterX.

Usage: `format_kv(kvs_dict, value_separator="<separator-character>", pair_separator="<separator-string>", quote_char="<quote-character>", always_quote=<boolean>)`

By default, `format_kv` uses `=` to separate values, and `, ` (comma and space) to separate the pairs:

```
filterx {
    ${MESSAGE} = format_kv(<input-dictionary>);
};
```

The `value_separator` option must be a single character, the `pair_separator` can be a string. For example, to use the colon (:) as the value separator and the semicolon (;) as the pair separator, use:

```
format_kv(<input-dictionary>, value_separator=":", pair_separator=";")
```

Quoting values

By default, AxoSyslog quotes only those values that contain a space, using double quotes (`"`). Any quote character inside such a value is escaped.

- `quote_char` : Available in AxoSyslog 4.28 and later.

  Sets the character used to quote values. Must be a single `"` or `'` character, other characters are rejected at configuration time. Default value: `"`

  For example, to quote values with apostrophes:

  ```
  format_kv(<input-dictionary>, quote_char="'")
  # Output: key1=value1, key2='it\'s'
  ```

- `always_quote`: Available in AxoSyslog 4.28 and later.

  If set to `true`, AxoSyslog quotes every value, not only the ones containing a space. Must be a boolean literal. Default value: `false`

```
format_kv({"key1":"value1","key2":42}, always_quote=true)
# Output: key1="value1", key2="42"
```

10.11.5 Log Event Extended Format (LEEF)

Format a FilterX dictionary into a LEEF 1.0 or 2.0 message with the format_leef() function.

Available in AxoSyslog 4.13 and later.

Formats a dictionary into the Log Event Extended Format (LEEF) . Both LEEF versions (1.0 and 2.0) are supported.

To parse LEEF messages, see Parse LEEF with FilterX.

Usage: `${MESSAGE} = format_leef(my_dictionary);`

For example:

```
my_dictionary =
{"leef_version":"1.0","vendor_name":"Microsoft","product_name":"MSExchange","produc
t_version":"4.0
SP1","event_id":"15345","src":"192.0.2.0","dst":"172.50.123.1","sev":"5",
"cat":"anomaly","srcPort":"81","dstPort":"21","usrName":"joe.black"};
```

Becomes:

```
LEEF:1.0|Microsoft|MSExchange|4.0 SP1|15345|src=192.0.2.0   dst=172.50.123.1
sev=5   cat=anomaly     srcPort=81      dstPort=21      usrName=joe.black
```

The following keys must be available in the dictionary, otherwise formatting fails with an error message like: `FILTERX ERROR; ....| format_leef(my_dictionary)', error='Failed to evaluate event formatter function:` .

- LEEF version 1: `leef_version` , `vendor_name` , `product_name` , `product_version` , `event_id` , `extensions`
- LEEF version 2: `leef_version` , `vendor_name` , `product_name` , `product_version` , `event_id` , `leef_delimiter` , `extensions`

10.11.6 RFC5424 syslog

Build an RFC5424 syslog message from FilterX data with format_syslog_5424(), and set its priority, timestamp, host, program, PID, and message ID.

Available in AxoSyslog 4.21 and later.

Formats data as an RFC5424 (IETF-syslog) syslog message.

Usage:

```
format_syslog_5424(
  message,
  add_octet_count=false,
  pri=<variable-or-expression>,
  timestamp=<variable-or-expression>,
  host=<variable-or-expression>,
  program=<variable-or-expression>,
  pid=<variable-or-expression>,
  msgid=<variable-or-expression>
)
```

Setting the `message` option is required. You can set the other options using any FilterX variable, function, or expression. If you specify a nonexisting variable, or if evaluating an expression fails, default values will be used.

If you want to include SDATA in the message, set the `SDATA` macro .

For example:

```
format_syslog_5424("My static message text", timestamp=datetime(1765146872.0),
host="host-value", program="prog-value", pid="5424", msgid="1234");
```

Becomes:

```
<13>1 2025-12-07T22:34:32.000000+00:00 host-value prog-value 5424 1234 - My static
message text
```

10.11.7 Windows Event Logs XML

Format a FilterX dictionary into Windows Event Log XML with format_windows_eventlog_xml(), a specialized version of format_xml().

Available in AxoSyslog 4.13 and later.

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Formats a dictionary into Windows Event Logs XML. It's a specialized version of the `format_xml()` function , all generic formatting tips apply to `format_windows_eventlog_xml()` as well.

To parse Windows Event Log XML, see Parse Windows Event Log XML with FilterX.

Usage: `format_windows_eventlog_xml(input_dictionary)`

Example usage:

```
$MESSAGE = format_windows_eventlog_xml({
    "Event": {
        "@xmlns": "http://schemas.microsoft.com/win/2004/08/events/event",
        "System": {
            "Provider": {"@Name": "EventCreate"},
            "EventID": {"@Qualifiers": "0", "#text": "999"},
            "Version": "0",
            "Level": "2",
            "Task": "0",
            "Opcode": "0",
            "Keywords": "0x80000000000000",
            "TimeCreated": {"@SystemTime": "2024-01-12T09:30:12.1566754Z"},
            "EventRecordID": "934",
            "Correlation": "",
            "Execution": {"@ProcessID": "0", "@ThreadID": "0"},
            "Channel": "Application",
            "Computer": "DESKTOP-2MBFIV7",
            "Security": {"@UserID": "S-1-5-21-3714454296-2738353472-899133108-
1001"},
        },
        "RenderingInfo": {
            "@Culture": "en-US",
            "Message": "foobar",
            "Level": "Error",
            "Task": "",
            "Opcode": "Info",
            "Channel": "",
            "Provider": "",
            "Keywords": {"Keyword": "Classic"},
        },
        "EventData": {"Data": ["foo", "bar"]},
    }
});
```

This JSON becomes the following XML:

```
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System>
<Provider Name='EventCreate'/><EventID Qualifiers='0'>999</EventID>
<Version>0</Version><Level>2</Level><Task>0</Task><Opcode>0</Opcode>
<Keywords>0x80000000000000</Keywords><TimeCreated SystemTime='2024-01-
12T09:30:12.1566754Z'/><EventRecordID>934</EventRecordID><Correlation/><Execution
ProcessID='0' ThreadID='0'/><Channel>Application</Channel><Computer>DESKTOP-
2MBFIV7</Computer><Security UserID='S-1-5-21-3714454296-2738353472-899133108-
1001'/></System><RenderingInfo Culture='en-US'><Message>foobar</Message>
<Level>Error</Level><Task/><Opcode>Info</Opcode><Channel/><Provider/><Keywords>
<Keyword>Classic</Keyword></Keywords></RenderingInfo><EventData><Data>foo</Data>
<Data>bar</Data></EventData></Event>
```

10.11.8 Format data as XML

Format a FilterX dictionary into XML with format_xml(), and see how attributes, text values, lists, and empty elements are mapped.

Available in AxoSyslog 4.13 and later.

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Formats a dictionary into XML. To format data as Windows Event Log XML, see Windows Event Logs XML.

To parse XML, see Parse XML with FilterX.

Usage: `format_xml({"key1":{"nestedkey":"value"}})`

The output XML for the previous example will be: `<key><nestedkey>value</nestedkey></key>`

Note the following points:

1. Store attributions in `@attr` key-value pairs.

```
JSON: {"foo": {"@bar": "123", "@baz": "bad"}}
XML:  <foo bar="123" baz="bad"/>
```

2. If an XML element has both attributes and a value, store text value under the `#text` key.

```
JSON: {"foo": {"@bar": "123", "#text": "baz"}}
XML:  <foo bar="123">baz</foo>
```

3. An XML element can have both a value and inner elements. We use the `#text` key here, too.

```
JSON: {"foo": {"#text": "bar", "baz": "123"}}
XML:  <foo>bar<baz>123</baz></foo>
```

4. JSON lists become values of separate tags:

```
JSON: {"a":{"b":["c","d"]}}
XML:  <a><b>c</b><b>d</b></a>
```

You can add attributions for specific elements of such lists:

```
JSON: {"a":{"b":["c",{"@attr":"attr_val","#text":"e"}]}}
XML:  <a><b>c</b><b attr='attr_val'>e</b></a>
```

5. A top-level JSON lists becomes a multi-root XML:

```
JSON: {"a":["b","c"]}
XML:  <a>b</a><a>c</a>
```

6. Numeric values become text:

```
JSON: {"a":100}
XML:  <a>100</a>
```

7. Empty elements are represented as short-format XML tags:

```
JSON: {"a":""}
XML:  <a/>
```

10.12 Handle and fix timezones and timestamps

Fix, guess, set, and inspect message timezones with the fix_timezone(), guess_timezone(), set_timezone(), and get_timezone_source() functions.

AxoSyslog FilterX has a few functions set or fix the timestamp and timezone of the messages.

fix_timezone

Available in AxoSyslog 4.24 and later. Similar to the `fix-time-zone()` rewrite operation .

Corrects the timezone of a message if it was parsed incorrectly for some reason, or if the client didn't include any timezone information in the message. For example:

```
datetime = strptime("2000-01-01T00:00:00 +0200", "%Y-%m-%dT%H:%M:%S %z");
timezone = "CET";
fixed_datetime = fix_timezone(datetime, timezone);
${MSG} = strftime("%Y-%m-%dT%H:%M:%S %z", fixed_datetime);
```

get_timezone_source

Available in AxoSyslog 4.24 and later.

Shows where the timezone information of the message originates from.

For example:

```
if (get_timezone_source(timestamp) === "assumed") {
  timestamp = fix_timezone(timestamp, "CET");
};
```

Possible values:

- `assumed` : The timestamp didn't contain timezone info, so AxoSyslog used the default (local) timezone
- `fixed` : The timezone was set using the `fix_timezone` or `set_timezone` functions
- `guessed` : The timezone info was set using `guess_timezone` function

- `parsed` : The timezone was parsed from the timestamp

guess_timezone

Available in AxoSyslog 4.24 and later. Similar to the `guess-time-zone()` rewrite operation .

Attempts to set the timezone of the message automatically, using heuristics on the timestamps. Normally AxoSyslog performs this operation automatically when it parses the incoming message. Use this function if you can't parse the incoming message for some reason, but you want to set the timezone automatically, for example, after you have preprocessed the message. Using this function is identical to using the `flags(guess-timezone)` flag in the source.

For example:

```
datetime = strptime("2000-01-01T00:00:00 +0200", "%Y-%m-%dT%H:%M:%S %z");
guessed_datetime = guess_timezone(datetime);
```

set_timezone

Available in AxoSyslog 4.24 and later. Similar to the `set-time-zone()` rewrite operation .

Sets the timezone of the message to a specific value, or converts an existing timezone to a different one. This operation is identical to setting the `time-zone()` option in the destination or as a global option, but can be applied selectively to the messages using conditions.

For example:

```
datetime = strptime("2000-01-01T00:00:00 +0200", "%Y-%m-%dT%H:%M:%S %z");
timezone = "CET";
set_datetime = set_timezone(datetime, timezone);
${MSG} = strftime("%Y-%m-%dT%H:%M:%S %z", set_datetime);
```

10.13 Handle SDATA in RFC5424 log records

Filter RFC5424 messages on their structured data with the get_sdata(), has_sdata(), and is_sdata_from_enterprise() FilterX functions.

Available in AxoSyslog 4.9 and later.

AxoSyslog FilterX has a few functions to handle the structured data (SDATA) part of RFC5424-formatted log messages . These functions allow you to filter messages based on their SDATA fields.

get_sdata()

Extracts the SDATA part of the message into a two-level dictionary, for example:

```
{"Originator@6876": {"sub": "Vimsvc.ha-eventmgr", "opID": "esxui-13c6-6b16"}}
```

```
filterx {
  sdata_json = get_sdata();
};
```

has_sdata()

Returns `true` if the SDATA field of the current message is not empty:

```
filterx {
  has_sdata();
};
```

is_sdata_from_enterprise

Filter messages based on enterprise ID in the SDATA field. For example:

```
filterx {
  is_sdata_from_enterprise("6876");
};
```

10.14 Handle OpenTelemetry log records

Route, modify, and create OpenTelemetry log records in FilterX, convert syslog messages to OTEL, and reference the otel_logrecord fields.

AxoSyslog allows you to process, manipulate, and create OpenTelemetry log messages using FilterX. For example, you can:

- route your OpenTelemetry messages to different destinations based on the content of the messages,
- change fields in the message (for example, add missing information, or delete unnecessary data), or
- convert incoming syslog messages to OpenTelemetry log messages.

The examples on this page map the incoming data to OTEL objects using the `${.otel_raw.*}` name-value pairs. This is what the `opentelemetry()` source creates by default, that is, in `mode(logmessage)` .

In AxoSyslog 4.28 and later, you can set `mode(filterx-dict)` in the source instead. In this case, the source directly creates the declared `log` , `resource` , and `scope` FilterX variables as plain dictionaries, so you can omit the input mapping step. Note that these variables are dictionaries, not OTEL objects, so the functions and the typed field handling described in this chapter don't apply to them.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Route OTEL messages

To route OTEL messages (such as the ones received through the `opentelemetry()` source) based on their content, configure the following:

1. Map the OpenTelemetry input message to OTEL objects in FilterX, so AxoSyslog handles their type properly. Add the following to your FilterX block:

```
log {
    source {opentelemetry()};
    filterx {
        # Input mapping
        declare log = otel_logrecord(${.otel_raw.log});
        declare resource = otel_resource(${.otel_raw.resource});
        declare scope = otel_scope(${.otel_raw.scope});
    };
    destination {
        # your opentelemetry destination settings
    };
};
```

2. Add FilterX statements that select the messages you need. The following example selects messages sent by the `nginx` application, received from the host called `example-host`.

```
log {
    source {opentelemetry()};
    filterx {
        # Input mapping
        declare log = otel_logrecord(${.otel_raw.log});
        declare resource = otel_resource(${.otel_raw.resource});
        declare scope = otel_scope(${.otel_raw.scope});

        # FilterX statements that act as filters
        resource.attributes["service.name"] == "nginx";
        resource.attributes["host.name"] == "example-host";
    };
    destination {
        # your opentelemetry destination settings
    };
};
```

For details on the common keys in log records, see the `otel_logrecord reference`.

Modify incoming OTEL

To modify messages received via the OpenTelemetry protocol (OTLP), such as the ones received using the `opentelemetry()` source , you have to configure the following:

1. Map the OpenTelemetry input message to OTEL objects in FilterX, so AxoSyslog handles their type properly. Add the following to your FilterX block:

```
log {
    source {opentelemetry()};
    filterx {
        # Input mapping
        declare log = otel_logrecord(${.otel_raw.log});
        declare resource = otel_resource(${.otel_raw.resource});
        declare scope = otel_scope(${.otel_raw.scope});
    };
    destination {
        # your opentelemetry destination settings
    };
};
```

2. After the mapping, you can access the elements of the different data structures as FilterX dictionaries , for example, the body of the message (`log.body`), its attributes (`log.attributes`), or the attributes of the resource (`resource.attributes`).

   The following example does two things:

   - It checks if the hostname resource attribute exists, and sets it to the sender IP address if it doesn't.

     ```
     if (not isset(resource.attributes["host.name"])) {
         resource.attributes["host.name"] = ${SOURCEIP};
     };
     ```

   - It checks whether the Timestamp field (which is optional) is set in the log object, and sets it to the date AxoSyslog received the message if it isn't.

```
if (log.observed_time_unix_nano == 0) {
    log.observed_time_unix_nano = ${R_UNIXTIME};
};
```

When inserted into the configuration, this will look like:

```
log {
    source {opentelemetry()};
    filterx {
        # Input mapping
        declare log = otel_logrecord(${.otel_raw.log});
        declare resource = otel_resource(${.otel_raw.resource});
        declare scope = otel_scope(${.otel_raw.scope});

        # Modifying the message
        if (not isset(resource.attributes["host.name"])) {
            resource.attributes["host.name"] = ${SOURCEIP};
        };
        if (log.observed_time_unix_nano == 0) {
            log.observed_time_unix_nano = ${R_UNIXTIME};
        };
    };
    destination {
        # your opentelemetry destination settings
    };
};
```

For details on mapping values, see the `otel_logrecord reference` .

3. Update the message with the modified objects so that your changes are included in the message sent to the destination:

```
log {
    source {opentelemetry()};
    filterx {
        # Input mapping
        declare log = otel_logrecord(${.otel_raw.log});
        declare resource = otel_resource(${.otel_raw.resource});
        declare scope = otel_scope(${.otel_raw.scope});

        # Modifying the message
        if (not isset(resource.attributes["host.name"])) {
            resource.attributes["host.name"] = ${SOURCEIP};
        };
        if (log.observed_time_unix_nano == 0) {
            log.observed_time_unix_nano = ${R_UNIXTIME};
        };

        # Update output
        ${.otel_raw.log} = log;
        ${.otel_raw.resource} = resource;
        ${.otel_raw.scope} = scope;
        ${.otel_raw.type} = "log";
    };
    destination {
        # your opentelemetry destination settings
    };
};
```

syslog to OTEL

To convert incoming syslog messages to OpenTelemetry log messages and send them to an OpenTelemetry receiver, you have to perform the following high-level steps in your configuration file:

1. Receive the incoming syslog messages.
2. Initialize the data structures required for OpenTelemetry log messages in a FilterX block .

3. Map the key-value pairs and macros of the syslog message to appropriate OpenTelemetry log record fields. There is no universal mapping scheme available, it depends on the source message and the receiver as well. For some examples, see the Example Mappings page in the OpenTelemetry documentation, or check the recommendations and requirements of your receiver. For details on the fields that are available in the AxoSyslog OTEL data structures, see the `otel_logrecord reference` .

   The following example includes a simple mapping for RFC3164-formatted syslog messages. Note that the body of the message is rendered as a string, not as structured data.

```
log {
    source {
    # Configure a source to receive your syslog messages
    };
    filterx {
        # Create the empty data structures for OpenTelemetry log records
        declare log = otel_logrecord();
        declare resource = otel_resource();
        declare scope = otel_scope();

        # Set the log resource fields and map syslog values
        resource.attributes["host.name"] = ${HOST};
        resource.attributes["service.name"] = ${PROGRAM};
        log.observed_time_unix_nano = ${R_UNIXTIME};
        log.body = ${MESSAGE};
        log.severity_number = ${LEVEL_NUM};

        # Update output
        ${.otel_raw.log} = log;
        ${.otel_raw.resource} = resource;
        ${.otel_raw.scope} = scope;
        ${.otel_raw.type} = "log";
    };
    destination {
        # your opentelemetry destination settings
    };
};
```

Log record fields

For the fields of the `otel_logrecord`, `otel_resource`, and `otel_scope` objects, see OpenTelemetry log record fields in FilterX.

10.14.1 OpenTelemetry log record fields in FilterX

Fields of the otel_logrecord, otel_resource, and otel_scope FilterX objects, with their types and severity values.

This page lists the fields of the OpenTelemetry objects you can use in FilterX. For examples of routing, modifying, and creating OpenTelemetry log records, see Handle OpenTelemetry log records.

otel_logrecord reference

OpenTelemetry log records can have the following fields. (Based on the official OpenTelemetry proto file .)

attributes

Type: `otel_kvlist`

Attributes that describe the event. Attribute keys MUST be unique.

body

The body of the log record. It can be a simple string, or any number of complex nested objects, such as lists and arrays.

flags

Type: `int`

Flags as a bit field.

observed_time_unix_nano

Type: `datetime`

The time when the event was observed by the collection system, expressed as nanoseconds elapsed since the UNIX Epoch (January 1, 1970, 00:00:00 UTC).

severity_number

Type: `int`

The severity of the message as a numerical value of the severity .

```
SEVERITY_NUMBER_UNSPECIFIED = 0;
SEVERITY_NUMBER_TRACE  = 1;
SEVERITY_NUMBER_TRACE2 = 2;
SEVERITY_NUMBER_TRACE3 = 3;
SEVERITY_NUMBER_TRACE4 = 4;
SEVERITY_NUMBER_DEBUG  = 5;
SEVERITY_NUMBER_DEBUG2 = 6;
SEVERITY_NUMBER_DEBUG3 = 7;
SEVERITY_NUMBER_DEBUG4 = 8;
SEVERITY_NUMBER_INFO   = 9;
SEVERITY_NUMBER_INFO2  = 10;
SEVERITY_NUMBER_INFO3  = 11;
SEVERITY_NUMBER_INFO4  = 12;
SEVERITY_NUMBER_WARN   = 13;
SEVERITY_NUMBER_WARN2  = 14;
SEVERITY_NUMBER_WARN3  = 15;
SEVERITY_NUMBER_WARN4  = 16;
SEVERITY_NUMBER_ERROR  = 17;
SEVERITY_NUMBER_ERROR2 = 18;
SEVERITY_NUMBER_ERROR3 = 19;
SEVERITY_NUMBER_ERROR4 = 20;
SEVERITY_NUMBER_FATAL  = 21;
SEVERITY_NUMBER_FATAL2 = 22;
SEVERITY_NUMBER_FATAL3 = 23;
SEVERITY_NUMBER_FATAL4 = 24;
```

severity_text

Type: `string`

The severity of the message as a string, one of:

```
"SEVERITY_NUMBER_TRACE"
"SEVERITY_NUMBER_TRACE2"
"SEVERITY_NUMBER_TRACE3"
"SEVERITY_NUMBER_TRACE4"
"SEVERITY_NUMBER_DEBUG"
"SEVERITY_NUMBER_DEBUG2"
"SEVERITY_NUMBER_DEBUG3"
"SEVERITY_NUMBER_DEBUG4"
"SEVERITY_NUMBER_INFO"
"SEVERITY_NUMBER_INFO2"
"SEVERITY_NUMBER_INFO3"
"SEVERITY_NUMBER_INFO4"
"SEVERITY_NUMBER_WARN"
"SEVERITY_NUMBER_WARN2"
"SEVERITY_NUMBER_WARN3"
"SEVERITY_NUMBER_WARN4"
"SEVERITY_NUMBER_ERROR"
"SEVERITY_NUMBER_ERROR2"
"SEVERITY_NUMBER_ERROR3"
"SEVERITY_NUMBER_ERROR4"
"SEVERITY_NUMBER_FATAL"
"SEVERITY_NUMBER_FATAL2"
"SEVERITY_NUMBER_FATAL3"
"SEVERITY_NUMBER_FATAL4"
```

span_id

Type: `bytes`

Unique identifier of a span within a trace, an 8-byte array.

time_unix_nano

Type: `datetime`

The time when the event occurred, expressed as nanoseconds elapsed since the UNIX Epoch (January 1, 1970, 00:00:00 UTC). If `0`, the timestamp is missing.

trace_id

Type: `bytes`

Unique identifier of a trace, a 16-byte array.

otel_resource reference

The resource describes the entity that produced the log record. It contains a set of attributes (key-value pairs) that must have unique keys. For example, it can contain the hostname and the name of the cluster.

otel_scope reference

Describes the instrumentation scope that sent the message. It may contain simple key-value pairs (strings or integers), but also arbitrary nested objects, such as lists and arrays. It usually contains a `name` and a `version` field.

10.15 Create metrics with FilterX

Count processed messages and maintain labeled metric counters with the update_metric() FilterX function and the metrics_labels type.

Available in AxoSyslog 4.9 and later.

You can use the `update_metric` function to count the processed messages, and create labeled metric counters based on the fields of the processed messages, similarly to the `metrics-probe()` parser . By default, `update_metric` increments the counter, but in AxoSyslog 4.28 and later you can use the `set` option to assign an absolute value to it instead.

You can configure the name of the counter to update and the labels to add. The name of the counter is an unnamed, mandatory option. Note that the name is automatically prefixed with the `syslogng_` string. For example:

```
update_metric(
    "my_counter_name",
    labels={
        "host": ${HOST},
        "app": ${PROGRAM},
        "id": ${SOURCE}
    }
);
```

This results in counters like:

```
syslogng_my_counter_name{app="example-app", host="localhost", source="s_local_1"} 3
```

update_metric options

increment

Type:	integer or variable
Default:	1

An integer, or an expression that resolves to an integer that defines the increment of the counter. The following example defines a counter called `syslogng_input_event_bytes_total`, and increases its value with the size of the incoming message (in bytes).

```
update_metric(
    "input_event_bytes_total",
    labels={
        "host": ${HOST},
        "app": ${PROGRAM},
        "id": ${SOURCE}
    },
    increment=${RAWMSG_SIZE}
);
```

The `increment` and `set` options are mutually exclusive, setting both is a configuration error.

labels

Type:	dict
Default:	`{}`

The labels used to create separate counters, based on the fields of the messages processed by `update_metric`. Use the following format:

```
labels(
    {
      "name-of-label1": "value-of-the-label1",
      ... ,
      "name-of-labelx": "value-of-the-labelx"
    }
)
```

level

Type:	integer (0-3)
Default:	0

Sets the stats level of the generated metrics.

> **Note**
>
> **Drivers configured with `internal(yes)` register their metrics on level 3. That way if you are creating an SCL, you can disable the built-in metrics of the driver, and create metrics manually using `update_metric`.**

set

Type:	non-negative integer or variable
Default:	N/A

Available in AxoSyslog 4.28 and later.

An integer, or an expression that resolves to an integer, that AxoSyslog assigns to the counter instead of incrementing it. Use this option to maintain gauge-like metrics, that is, metrics that can go up and down. The following example sets the counter called `syslogng_demo_gauge` to the value reported in the message:

```
update_metric(
    "demo_gauge",
    labels={
        "host": ${HOST}
    },
    set=int(${MESSAGE})
);
```

The `set` and `increment` options are mutually exclusive, setting both is a configuration error.

Only non-negative values are supported. If the value is negative, AxoSyslog leaves the counter unchanged and logs an error, for example:

```
FilterX: update_metric() eval error, skipping; err_idx='[1/1]', expr='n/a',
error='Failed to evaluate update_metric(): Metric value must be non-negative, got:
-100'
```

metrics_labels

Available in AxoSyslog 4.10 and later.

`metrics_labels` is a dict-like data type to store metric labels directly. You can use the `metrics_labels` function to convert key-values to metric labels directly . This is useful when you have multiple `update_metric()` function calls, because it avoids re-rendering the labels, greatly improves performance.

The stored labels are sorted alphabetically, but note that key collisions are not detected. You can use the `dedup_metrics_labels()` function to deduplicate labels. However, this takes CPU time, it's better to avoid inserting keys multiple times.

10.16 Update to FilterX

Convert existing filter expressions, rewrite rules, and parsers in your configuration to equivalent FilterX statements and functions.

The following sections show you how you can change your existing filters, parsers, and rewrite rules to FilterX statements. Note that:

- Many examples in the FilterX documentation were adapted from the existing filter, parser, and rewrite examples to show how you can achieve the same functionality with FilterX.
- Don't worry if you can't update something to FilterX. While you can't use other blocks within a FilterX block, you can use both in a log statement, for example, you can use a FilterX block, then a parser if needed.
- There is no push to use FilterX. You can keep using the traditional blocks if they satisfy your requirements.

Update filters to FilterX

This section shows you how to update your existing `filter` expressions to `filterx`.

You can replace most filter functions with a simple value comparison of the appropriate macro, for example:

- `facility(user)` with `${FACILITY} == "user"`
- `host("example-host")` with `${HOST} == "example-host"`
- `inlist()` with the `in` list membership operator
- `level(warning)` with `${LEVEL} == "warning"`

  If you want to check for a range of levels, use numerical comparison with the `${LEVEL_NUM}` macro instead. For a list of numerical level values, see LEVEL_NUM.
- `message("example")` with `${MESSAGE} =~ "example"` (see the equal tilde operator for details)
- `netmask()` and `netmask6()` with `subnet` and a list membership check, for example, `netmask(192.168.5.0/255.255.255.0)` becomes `${SOURCEIP} in subnet("192.168.5.0/255.255.255.0");`. For details, see IP addresses and subnets.
- `program(nginx)` with `${PROGRAM} == "nginx"`
- `source(my-source)` with `${SOURCE} == "my-source"`

You can compare values and use boolean operators similarly to filters.

Since all FilterX statements must match a message to pass the FilterX block, you can often replace complex boolean filter expressions with multiple, simple FilterX statements. For example, consider the following filter statement:

```
filter { host("example1") and program("nginx"); };
```

The following is the same FilterX statement:

```
filterx { ${HOST} == "example1" and ${PROGRAM} == "nginx"; };
```

which is equivalent to:

```
filterx {
    ${HOST} == "example1";
    ${PROGRAM} == "nginx";
};
```

The following filter functions have no equivalents in FilterX yet:

- The `filter()` filter function . You can't call a FilterX block from another FilterX block, but you can access name-value pairs and pass variables from multiple FilterX blocks.
- `rate-limit()`
- `tags()`

Update rewrite rules

This section shows you how to update your existing `rewrite` expressions to `filterx`.

You can replace most rewrite rules with FilterX functions and value assignments, for example:

- `rewrite{subst()}` with the `regexp_subst` FilterX function
- `rewrite{set()}` with value assignments
- `rewrite{unset()}` with the `unset` FilterX function

- `rewrite{rename()}` with assigning a value to the new field, then using the `unset` function on the old field
- Timezone manipulation with the similar FilterX functions .
- `set-pri()` , `set-severity()` , and `set-facility()` with the `set_pri` FilterX function
- Setting multiple fields at once with the `set_fields` FilterX function
- Conditional rewrites with value comparisons in the FilterX block. For an example, see FilterX use cases and examples.

The following rewrite rules have no equivalents in FilterX yet:

- `credit-card-mask()` and `credit-card-hash()` . You can mask the numbers with the `regexp_subst` FilterX function instead.
- `set-tag()` and `clear-tag()`
- `set-matches()`

Update parsers

This section shows you how to update your existing `parser` expressions to `filterx` .

You can replace most parsers with FilterX functions. These functions return the parsed data, which you can assign to a variable or a name-value pair. For example, you can replace:

- `csv-parser()` with the `parse_csv` FilterX function
- `date-parser()` with the `strptime` FilterX function. To set the timestamp of the message, use `set_timestamp` on the result.
- `json-parser()` with the `json` FilterX function
- `kv-parser()` with the `parse_kv` FilterX function
- `metrics-probe()` with the `update_metric` FilterX function
- `regexp-parser()` with the `regexp_search` FilterX function
- `windows-eventlog-xml-parser()` with the `parse_windows_eventlog_xml` FilterX function
- `xml-parser()` with the `parse_xml` FilterX function

To parse CEF and LEEF messages, use the `parse_cef` and `parse_leef` FilterX functions. These formats have no classic parsers.

The following parsers have no equivalents in FilterX yet. You can still use them in the same log path as your FilterX blocks.

- `db-parser()` (pattern databases)
- `grouping-by()` and other correlation parsers

- Application-specific parsers, like the Cisco, FortiGate, or iptables parsers

10.17 FilterX operator reference

Reference of FilterX operators: arithmetic, comparison, boolean, null coalescing, assign if non-null, list membership, regexp match, slicing, ternary.

This page describes the operators you can use in FilterX blocks .

Arithmetic operators

Available in AxoSyslog 4.12 and later.

The `+` (addition), `-` (substraction), `*` (multiplication), `/` (division), and `%` (modulo) operators allow you to perform arithmetic operations on numeric (integer or double) values. For example:

```
if (3 - 3 == 0) { ... }
if (3.0 * 3 == 9.0) { ... }
if (3.0 / 3.0 == 1.0) { ... }
if (4 % 3 == 1) { ... }
```

Note that:

- The `%` operator only accepts integer values.
- If one operand is integer and the other is double, the result will be double.
- The `+` operator can add strings and other types as well, for details, see the Plus operator .
- To increase the value of a variable, see the Plus equal operator .
- In version 4.18 and later, you can use the `+` and `-` operators as unary operators with a single operand to indicate a positive or a negative value. For example:

```
a = 42;
b = -a;
# b is -42
```

Plus operator

The plus operator (`+`) adds two arguments, if possible. (For example, you can't add two datetime values.)

- You can use it to add two numbers (two integers, two double values). If you add a double to an integer, the result is a double.
- Adding two strings concatenates the strings. Note that if you want to have spaces between the added elements, you have to add them manually, like in Python, for example:

```
${MESSAGE} = ${HOST} + " first part of the message," + " second part of the message" + "\n";
```

- Adding two lists merges the lists. Available in AxoSyslog 4.9 and later.
- Adding two dicts updates the dict with the values of the second operand. For example:

```
x = {"key1": "value1", "key2": "value1"};
y = {"key3": "value1", "key2": "value2"};
${MESSAGE} = x + y; # ${MESSAGE} value is {"key1": "value1", "key3": "value1", "key2": "value2"};
```

  Available in AxoSyslog 4.9 and later.

Plus equal operator

The `+=` operator increases the value of a variable with the value on the right. Exactly how the addition happens depends on the type of the variable.

- For numeric types (`int` and `double`), the result is the sum of the values. For example:

```
a = 3;
a += 4;
# a is 7

b = 3.3;
b += 4.1;
# b is 7.4
```

Adding a double value to an integer changes the integer into a double:

```
c = 3;
c += 4.1;
# c is 7.1 and becomes a double
```

- For strings (including string values in an object), it concatenates the strings. For example:

```
mystring = "axo";
mystring += "flow";
# mystring is axoflow
```

- For lists, it appends the new values to the list. For example:

```
mylist = json_array(["one", "two"]);
mylist += ["let's", "go"];
# mylist is ["one", "two", "let's", "go"]
```

- For datetime variables, it increments the time. Note that you can add only integer and double values to a datetime, and:
    - When adding an integer, it must be the number of microseconds you want to add. For example:

```
d = strptime("2000-01-01T00:00:00Z", "%Y-%m-%dT%H:%M:%S%z");
d += 3600000000; # 1 hour in microseconds
# d is "2000-01-01T01:00:00.000+00:00"
```

- When adding a double, the integer part must be the number of seconds you want to add. For example:

```
d = strptime("2000-01-01T00:00:00Z", "%Y-%m-%dT%H:%M:%S%z");
d += 3600.000; # 3600 seconds, 1 hour
# d is "2000-01-01T01:00:00.000+00:00"
```

Comparison operators

Comparison operators allow you to compare values of macros, variables, and expressions as numbers (`==, <, <=, >=, >, !=`) or as strings (`eq, lt, le, gt, ge, ne`). You can also check for type equality (`===`) and strict inequality (`!==`). For details and examples, see Comparing values in FilterX.

Boolean operators

The `not` , `or` , `and` operators allow you to combine any number of comparisons and expressions. For details and examples, see Boolean operators in FilterX.

Assign if non-null (=??) operator

Available in AxoSyslog 4.10 and later.

Assigns the right operand to the left operand if the right operand exists and is not null. Note that evaluation errors of the right-hand operand will be suppressed.

```
left-operand =?? right-operand
```

For example:

```
resource.attributes['service.name'] =?? $PROGRAM;
```

Using the `=??` operator is equivalent to the following expression, but using `=??` has better performance.

```
if (isset($PROGRAM) ?? false) {
    resource.attributes['service.name'] = $PROGRAM;
};
```

Create dict element if non-null (:??) operator

Available in AxoSyslog 4.15 and later.

Creates the dict element in the left operand with the value of the right operand if the right operand exists and is not null. Note that evaluation errors of the right-hand operand will be suppressed.

For example, the following dict will have only one element, the `good-field`:

```
my_dict = {
    "skipped-because-null":?? null,
    "skipped-because-error":?? nonexistingvariable,
    "good-field": "static-value"
};
```

Null coalescing operator

The null coalescing operator returns the result of the left operand if it exists and is not null, otherwise it returns the operand on the right.

```
left-operand ?? right-operand
```

You can use it to define a default value, or to handle errors in your FilterX statements: if evaluating the left-side operand returns an error, the right-side operand is evaluated instead.

For example, if a key of a JSON object doesn't exist for every message, you can set it to a default value:

```
${MESSAGE} = json["BODY"] ?? "Empty message"
```

List membership operator

Available in AxoSyslog 4.12 and later.

You can check whether a value is in a list using the `in` and `not in` membership operators. If you're checking the value of a variable that can have another type than the elements of a list, explicitly convert the value to the correct type (for example, string) before the comparison.

```
my_array = json_array(['hostname_one', 'hostname_two', 'hostname_three']);
if (${HOST} in my_array) {
    # ...
}
```

Starting with AxoSyslog 4.22, you can also use the operators to check membership in `dict` keys. For example:

```
my_dict = {"key-1": "value", "key-2": "another value"};
my_search_key = "key-1";
if (my_search_key in my_dict) {
    ${MSG} = "Found: " + my_dict[my_search_key];
} else {
    ${MSG} = "Not Found";
};
```

The `in` operator can be used to replicate the functionality of the in-list() filter function in FilterX. If you want to populate the list from a file, use the cache_json_file() FilterX function.

To check if an IP address is a member of a subnet, see IP addresses and subnets.

Regexp match (equal tilde)

To check if a value contains a string or matches a regular expression, use the `=~` operator. For example, the following statement is true if the `${MESSAGE}` contains the word `error`:

```
${MESSAGE} =~ "error";
```

Use the `!~` operator to check if a literal string or variable doesn't contain an expression. For example, the following statement is true if the `${MESSAGE}` doesn't contain the word `error`:

```
${MESSAGE} !~ "error";
```

> **Note**
>
> - If you want to process the matches of a search, use the regexp_search FilterX function.
> - If you want to rewrite or modify the matches of a search, use the regexp_subst FilterX function.

Note the following points:

- Regular expressions are case sensitive by default. For case insensitive matches, add `(?i)` to the beginning of your pattern.
- You can use regexp constants (slash-enclosed regexps) within FilterX blocks to simplify escaping special characters, for example, `/^beginning and end$/`.
- FilterX regular expressions are interpreted in "leave the backslash alone mode", meaning that a backslash in a string before something that doesn't need to be escaped and will be interpreted as a literal backslash character. For example, `string\more-string` is equivalent to `string\\more-string`.

String slicing (..)

Available in AxoSyslog 4.15 and later.

You can slice strings at the specified index using the `..` operator to get a section of the string. Indexing starts at 0. You can omit the index to refer to the beginning or the end of the string. For example:

```
filterx {
  str = "example";
  idx = 3;
  my_string = str[idx..5];
  # Value of my_string is "mp";

  my_string = str[..idx];
  # Value of my_string is "exa";

  my_string = str[idx..];
  # Value of my_string is "mple";
};
```

Staring with AxoSyslog version 4.17, you can use negative indexes to refer to characters from the end of the string, for example:

```
filterx {
  str = "example";
  str[..-2] == "examp";
  str[-3..] == "ple";
  str[2..-2] == "amp";
};
```

Staring with AxoSyslog version 4.18, you can use the `-` operator with a variable to to refer to characters from the end of the string, for example:

```
filterx {
  str = "example";
  idx = 3;
  str[..-idx] == "exam";
  str[-idx..] == "ple";
  str[idx..-idx] == "m";
};
```

Ternary conditional operator

The ternary conditional operator evaluates an expression and returns the first argument if the expression is true, and the second argument if it's false.

Syntax:

```
<expression> ? <return-if-true> : <return-if-false>
```

For example, the following example checks the value of the `${LEVEL_NUM}` macro and returns `low` if it's lower than 5, `high` otherwise.

```
(${LEVEL_NUM} < 5 ) ? "low" : "high";
```

You can also use it to check if a value is set, and set it to a default value if it isn't, but for this use case we recommend using the Null coalescing operator :

```
${HOST} = isset(${HOST}) ? ${HOST} : "default-hostname"
```

10.18 FilterX function reference

Reference of all FilterX functions, from base64_decode() to vars(), with their arguments and usage examples.

This page describes the functions you can use in FilterX blocks .

Functions have arguments that can be either mandatory or optional.

- Mandatory options are always positional, so you need to pass them in the correct order. You cannot set them in the `arg=value` format.
- Optional arguments are always named, like `arg=value`. You can pass optional arguments in any order.

base64_decode

Available in AxoSyslog 4.25 and later.

Decodes a Base64-encoded string and returns the result as a bytes value.

Usage: `base64_decode(string)`

For example:

```
base64_decode("Zm9vYmFy");  # Returns the bytes "foobar"
base64_decode(base64_encode("szilvafa"));  # Round-trips back to the original value
```

base64_encode

Available in AxoSyslog 4.25 and later.

Encodes a string or bytes value as a Base64 string.

Usage: `base64_encode(string_or_bytes)`

For example:

```
base64_encode("foobar");  # Returns "Zm9vYmFy"
```

bool

Cast a value into a boolean. The result is the truthiness of the value: `true` for truthy values, `false` for falsy values.

Usage: `bool(<value>)`

For example:

```
has_tags = bool(tags);  # false if tags is an empty list
```

Note that the string is not parsed: `bool("false")` returns `true`, because `"false"` is a non-empty string. To check a string, compare it instead, for example: `${.enabled} == "true"`.

bytes

Cast a string or a `protobuf` value into `bytes`, a type that represents binary data. Casting any other type causes an error.

Usage: `bytes(<string or protobuf value>)`

For example:

```
raw = bytes(${MESSAGE});
```

Functions that decode binary data, like `base64_decode` and `hex_decode`, return `bytes` values.

cache_json_file

Load the contents of an external JSON file in an efficient manner. You can use this function to lookup contextual information. (Basically, this is a FilterX-specific implementation of the `add-contextual-data() functionality`.)

AxoSyslog automatically detects if the file is updated and reloads the file.

Usage: `cache_json_file("/path/to/file.json", default_value=<fallback-value>)`

Parameters:

- `"/path/to/file.json"` : Path to the JSON file to load. (Mandatory)
- `default_value` : (Optional) Available in AxoSyslog 4.25 and later.

  A fallback value to return when the file doesn't exist or fails to load (for example, due to a parse error).

  > **CAUTION:**
  >
  > Without the `default_value` argument, AxoSyslog fails to start if the file is unavailable.

  Note that the `default_value` is only used until AxoSyslog successfully loads at least one version of the JSON file. For example:

  1. The file doesn't exist at startup: AxoSyslog uses `default_value` .
  2. Later the file becomes available: AxoSyslog uses the content of file content.
  3. The file is updated but invalid: AxoSyslog keeps using the old version of the file, not the `default_value` .

For example, if your `context-info-db.json` file contains the following:

```
{
  "nginx": "web",
  "httpd": "web",
  "apache": "web",
  "mysql": "db",
  "postgresql": "db"
}
```

Then the following FilterX expression selects only "web" traffic:

```
filterx {
  declare known_apps = cache_json_file("/context-info-db.json");
  ${app} = known_apps[${PROGRAM}] ?? "unknown";
  ${app} == "web";  # drop everything that's not a web server log
}
```

To avoid failures when the file might not yet exist, provide a `default_value` :

```
filterx {
  declare known_apps = cache_json_file("/context-info-db.json", default_value=
{"nginx": "web","postgresql": "db"});
  ${app} = known_apps[${PROGRAM}] ?? "unknown";
}
```

datetime

Cast a value into a datetime variable.

Usage: `datetime(<string or expression to cast as datetime>)`

For example:

```
date = datetime("1701350398.123000+01:00");
```

Usually, you use the strptime FilterX function to create datetime values. Alternatively, you can cast an integer, double, string, or isodate variable into datetime with the `datetime()` FilterX function. Note that:

- When casting from an integer, the integer is the number of microseconds elapsed since the UNIX epoch (00:00:00 UTC on 1 January 1970).
- When casting from a double, the double is the number of seconds elapsed since the UNIX epoch (00:00:00 UTC on 1 January 1970). (The part before the floating points is the seconds, the part after the floating point is the microseconds.)
- When casting from a string, the string (for example, `1701350398.123000+01:00`) is interpreted as: `<the number of seconds elapsed since the UNIX epoch>.<microseconds>+<timezone relative to UTC (GMT +00:00)>`

dedup_metrics_labels

Deduplicate `metrics_labels` objects. For details, see metrics_labels.

dict

Available in AxoSyslog 4.10 and later.

Create a dict. `dict()` is the same as `json()` . If the argument is a JSON string, it's parsed like with `parse_json` . Without an argument, it returns an empty dict, the same as `{}` .

Usage: `dict(<value>)`

For example:

```
d = dict('{"host": "web01", "port": 443}');
d.port == 443;
```

dict_to_pairs

Convert dicts to list of pairs.

Usage: `dict_to_pairs(<input-dict>, <key-name>, <value-name>)`

```
my_dict = {
    "key_1": "value_1",
    "key_2": "value_2",
    "key_3": ["value_3", "value_4"],
};

my_list = dict_to_pairs(my_dict, "key", "value");
# The value of my_list will be:
# [
#   {"key":"key_1","value":"value_1"},
#   {"key":"key_2","value":"value_2"},
#   {"key":"key_3","value":["value_3","value_4"]}
# ]
```

digest

Available in AxoSyslog 4.25 and later.

Computes a cryptographic hash of a string or bytes value and returns the raw hash as a bytes object. Use the optional `alg=` argument to select the hash algorithm. If `alg=` is not set, the default is `sha256` . The algorithm name is passed to OpenSSL, so any name

accepted by `EVP_get_digestbyname` is supported (for example, `md5`, `sha1`, `sha256`, `sha512`). An unknown algorithm name causes a configuration error.

To obtain the hash as a hexadecimal string instead of bytes, use the convenience functions `md5`, `sha1`, `sha256`, or `sha512`. Usage: `digest(string_or_bytes, alg="sha256")`

For example:

```
digest("foobar");                   # Raw SHA-256 hash as bytes
digest("foobar", alg="md5");        # Raw MD5 hash as bytes
```

double

Cast a value into a double (floating-point number).

Usage: `double(<value>)`

- Integers and booleans are converted to a double.
- Strings are parsed as a decimal number. The whole string must be a number, otherwise the cast fails.
- A `datetime` value becomes the number of seconds elapsed since the UNIX epoch, with the microseconds as the fraction.
- Any other type causes an error.

For example:

```
${.response_time} = double(${.apache.response_time_str});
```

dpath

Available in AxoSyslog 4.17 and later.

Assigns a value to a dictionary and creates any elements of the path that don't exist. For example:

```
js = json({
"key1": "one",
"key2": "two",
"key3": "three"
});

dpath(js.key4.key41.key412) = "nested value"
```

The value of the dictionary will be:

```
js = json({
"key1": "one",
"key2": "two",
"key3": "three",
"key4": {
    "key41": {
        "key412": "nested value"
        }
    }
});
```

endswith

Available in AxoSyslog 4.9 and later.

Returns true if the input string ends with the specified substring. By default, matches are case sensitive. Usage:

```
endswith(input-string, substring);
endswith(input-string, [substring_1, substring_2], ignorecase=true);
```

For details, see String search in FilterX.

failure_info, failure_info_clear, failure_info_enable, failure_info_meta

Available in AxoSyslog 4.12 and later.

Collect and return information about failing FilterX expressions. For details, see Track failures.

flatten

Flattens the nested elements of an object using the specified separator, similarly to the `format-flat-json()` template function . For example, you can use it to flatten nested JSON objects in the output if the receiving application cannot handle nested JSON objects.

Usage: `flatten(dict_or_list, separator=".")`

You can use multi-character separators, for example, `=>` . If you omit the separator, the default dot (`.`) separator is used.

```
sample-dict = json({"a": {"b": {"c": "1"}}});
${MESSAGE} = flatten(sample-dict);
```

The value of `${MESSAGE}` will be: `{"a.b.c": "1"}`

fix_timezone

See fix_timezone.

format_cef

Available in AxoSyslog 4.13 and later.

Formats a dictionary into the Common Event Format (CEF) .

Usage: `${MESSAGE} = format_cef(my_dictionary);`

For details, see Common Event Format (CEF).

format_csv

Formats a dictionary or a list into a comma-separated string.

Usage: `format_csv(<input-list-or-dict>, columns=<json-list>, delimiter=<delimiter-character>, default_value=<string>)`

For details, see Format data as comma-separated values.

format_isodate

Available in AxoSyslog 4.23 and later.

Formats a date as ISODATE: `%Y-%m-%dT%H:%M:%S%z`. For example:

```
my_date = strptime("2000-01-02T03:04:05.678901-07:00", "%Y-%m-%dT%H:%M:%S.%f%Z");
${MESSAGE} = format_isodate(my_date);
# The value of ${MESSAGE} is 2000-01-02T03:04:05.678901-07:00
```

format_json

Formats any value into a raw JSON string.

Usage: `format_json($data)`

For details, see JSON.

format_kv

Formats a dictionary into a string containing key=value pairs.

Usage: `format_kv(kvs_dict, value_separator="<separator-character>", pair_separator="<separator-string>", quote_char="<quote-character>", always_quote=<boolean>)`

The `quote_char` and `always_quote` options are available in AxoSyslog 4.28 and later.

For details, see Key-value pairs.

format_leef

Available in AxoSyslog 4.13 and later.

Formats a dictionary into the Log Event Extended Format (LEEF) . Both LEEF versions (1.0 and 2.0) are supported.

Usage: `${MESSAGE} = format_leef(my_dictionary);`

For details, see Log Event Extended Format (LEEF).

format_syslog_5424

Available in AxoSyslog 4.21 and later.

Formats data as an RFC5424 (IETF-syslog) syslog message.

For details, see RFC5424 syslog.

format_xml

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Available in AxoSyslog 4.13 and later.

Formats a dictionary into XML.

Usage: `format_xml({"key1":{"nestedkey":"value"}})`

For details, see Format data as XML.

format_windows_eventlog_xml

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Available in AxoSyslog 4.13 and later.

Formats a dictionary into Windows Event Logs XML. It's a specialized version of the `format_xml()` function , all generic formatting tips apply to `format_windows_eventlog_xml()` as well.

Usage: `format_windows_eventlog_xml(input_dictionary)`

For details, see Windows Event Logs XML.

get_sdata

See Handle SDATA in RFC5424 log records.

get_timestamp

Available in AxoSyslog 4.12 and later.

Usage: `get_timestamp(stamp="stamp")`

Returns a timestamp of the message as a `datetime` value. The `stamp` argument determines which timestamp to return:

- `stamp` : The timestamp of the message (the date in the message header, or the date when AxoSyslog received the message if the message has no date). This is the default.
- `recvd` : The date when AxoSyslog received the message.

The `stamp` argument must be a literal string. Any other value causes a configuration error.

For example:

```
received = get_timestamp(stamp="recvd");
${.processing_delay_us} = int(received) - int(get_timestamp());
```

Casting a `datetime` value to integer returns the number of microseconds elapsed since the UNIX epoch. To get milliseconds, divide it by 1000: `int(get_timestamp()) / 1000` .

To change a timestamp of the message, use `set_timestamp` .

get_timezone_source

See get_timezone_source.

glob_match

Available in AxoSyslog 4.25 and later.

Matches a filename (or any string) against one or more glob patterns and returns `true` if the filename matches any of the patterns, `false` otherwise. Note that `/` separators are matched literally (not by wildcards), and a leading `.` must be matched explicitly.

Usage: `glob_match(filename, patterns)`

- `filename` : The string to test.
- `patterns` : A single pattern string, or a list of pattern strings. When a list is provided, the function returns `true` as soon as any pattern matches.

Using a non-string `filename`, or a non-string element in the `patterns` list, causes a runtime error.

For example:

```
glob_match("filename.log", "*.log");                # Returns true
glob_match("filename.log", ["*.log", "*.txt"]);     # Returns true
glob_match("filename.cfg", ["*.log", "*.txt"]);     # Returns false
glob_match("/var/log/syslog", ["/var/log/*"]);      # Returns true
```

guess_timezone

See guess_timezone.

has_sdata

Returns `true` if the SDATA field of the current message is not empty. See Handle SDATA in RFC5424 log records.

hex_decode

Available in AxoSyslog 4.25 and later.

Decodes a lowercase or uppercase hexadecimal string and returns the result as bytes. The input length must be even, and each character must be a valid hexadecimal digit. Invalid characters or an odd-length input cause a runtime error.

Usage: `hex_decode(string)`

For example:

```
hex_decode("666f6f626172");          # Returns the bytes "foobar"
```

hex_encode

Available in AxoSyslog 4.25 and later.

Encodes a string or bytes value as a lowercase hexadecimal string.

Usage: `hex_encode(string_or_bytes)`

For example:

```
hex_encode("foobar");  # Returns "666f6f626172"
```

includes

Available in AxoSyslog 4.9 and later.

Returns true if the input string contains the specified substring. By default, matches are case sensitive. Usage:

```
includes(input-string, substring);
includes(input-string, [substring_1, substring_2], ignorecase=true, limit=10);
```

For details, see String search in FilterX.

int

Cast a value into an integer.

Usage: `int(<value>)`

- Doubles are rounded to the nearest integer, for example, `int(2.5)` is `3`. Values outside the 64-bit integer range are clamped to the smallest or largest integer.
- Booleans become `0` or `1`.
- Strings are parsed as a base-10 integer. The whole string must be a number, otherwise the cast fails: `int("42")` works, `int("42ms")` and `int("4.2")` fail.
- A `datetime` value becomes the number of microseconds elapsed since the UNIX epoch. To get milliseconds, divide it by 1000: `int(get_timestamp()) / 1000`. Dividing two integers returns an integer.
- Any other type causes an error.

For example:

```
${.status} = int(${.apache.status});
```

If the cast can fail, use the null coalescing operator to set a default value: `${.status} = int(${.apache.status}) ?? 0;`

ip

See ip.

isodate

Parses a string as a date in ISODATE format: `%Y-%m-%dT%H:%M:%S%z`

is_sdata_from_enterprise()

See Handle SDATA in RFC5424 log records.

isset

Returns true if the argument exists and its value is not empty or null.

Usage: `isset(<name of a variable, macro, or name-value pair>)`

If you want to assign a value to a variable if it's not set, use the `=??` operator .

istype

Returns true if the object (first argument) has the specified type (second argument). The type must be a quoted string. (See List of type names .)

Usage: `istype(object, "type_str")`

For example:

```
obj = json();
istype(obj, "json_object"); # True

istype(${PID}, "string");
istype(my-local-json-object.mylist, "json_array");
```

If the object doesn't exist, `istype()` returns with an error, causing the FilterX statement to become false, and logs an error message to the `internal()` source of AxoSyslog.

json

Cast a value into a JSON object.

Usage: `json(<string or expression to cast to json>)`

For example:

```
js_dict = json({"key": "value"});
```

Starting with version 4.9, you can use `{}` without the `json()` keyword as well. For example, the following creates an empty JSON object:

```
js_dict = {};
```

json_array

Cast a value into a JSON array.

Usage: `json_array(<string or expression to cast to json array>)`

For example:

```
js_list = json_array(["first_element", "second_element", "third_element"]);
```

Starting with version 4.9, you can use `[]` without the `json_array()` keyword as well. For example, the following creates an empty JSON list:

```
js_dict = [];
```

keys

Returns the top-level keys of a dictionary. This provides a simple way to inspect or iterate over the immediate keys without traversing the structure. The `keys()` function:

- Returns a list of dictionary keys as an array.
- Includes only the top-level keys, ignoring nested structures.
- The resulting array supports immediate indexing for quick key retrieval.
- When called on an empty dictionary, `keys` returns an empty dictionary (`[]`).

For example:

```
dict = {"level1-key1":{"level2-key1":{"level3-key1":"value1"}},"level1-key2":
{"level2-key2":{"level3-key2":"value2"}}};

# accessing the top level, returns: ["level1-key1", "level1-key2"]
a = keys(dict);

# accessing nested levels directly, returns: ["level2-key1"]
b = keys(dict["level1-key1"]);

# directly index the result of keys() to access specific keys is possible, returns:
["level1-key1"])
c = keys(dict)[0];
```

len

Returns the number of items in an object as an integer: the length (number of characters) of a string, the number of elements in a list, or the number of keys in an object.

Usage: `len(object)`

list

Available in AxoSyslog 4.12 and later.

Create a list. `list()` is the same as `json_array()` . If the argument is a JSON string, it's parsed like with `parse_json` . Without an argument, it returns an empty list, the same as `[]` .

Usage: `list(<value>)`

For example:

```
ports = list("[80, 443]");
```

load_vars

Loads variables from a dict. It's the inverse of `vars()` . It loads and declares FilterX-level variables. If a key in the dict begins with the `$` character, it's loaded as an AxoSyslog macro. This function can be used to transfer several variables between FilterX blocks on different log paths and messages.

lower

Converts all characters of a string lowercase characters.

Usage: `lower(string)`

md5

Available in AxoSyslog 4.25 and later.

Computes the MD5 hash of a string or bytes value and returns the result as a lowercase hexadecimal string. To obtain the raw hash as bytes, use the `digest` function with `alg="md5"` .

Usage: `md5(string_or_bytes)`

For example:

```
md5("foobar");  # Returns "3858f62230ac3c915f300c664312c63f"
```

metrics_labels

Convert key-values to metric labels directly. For details, see metrics_labels.

move

Available in AxoSyslog 4.23 and later.

Moves the specified variable to its new location, instead of copying it. This is equivalent to using a value assignment and an unset function, but has better performance. The following example moves the JSON object `old` into the `new.nest` field:

```
filterx {
  old = {"key1":"value1", "key2":"value2"};
  new = {"foo":"foovalue", "bar":"barvalue"};

  new.nest = move(old);
};
```

otel_array

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Creates a dictionary represented as an OpenTelemetry array.

otel_kvlist

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Creates a dictionary represented as an OpenTelemetry key-value list.

otel_logrecord

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Creates an OpenTelemetry log record object .

otel_resource

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Creates an OpenTelemetry resource object .

otel_scope

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Creates an OpenTelemetry scope object .

parse_cef

Parse messages formatted in the Common Event Format (CEF) .

Usage: `parse_cef(<input-string>, value_separator="=", pair_separator="|", separate_extensions=false)`

For details, see Parse CEF with FilterX.

parse_csv

Split a comma-separated or similar string.

Usage: `parse_csv(msg_str [columns=json_array, delimiter=string, string_delimiters=json_array, dialect=string, strip_whitespace=boolean, greedy=boolean])`

For details, see Parse comma-separated values with FilterX.

parse_json

Available in AxoSyslog 4.11 and later.

Parse a JSON string, and return the value it represents.

Usage: `parse_json(<string>)`

Unlike `json()` , which accepts only a JSON object, `parse_json()` accepts any JSON value, and returns a value of the matching type: a dict, a list, a string, a number, a boolean, or `null` .

For example:

```
js = parse_json(${MESSAGE});
js.data_map.key1 == 1;
```

If the argument isn't a string, or it isn't valid JSON, `parse_json()` returns an error. To create JSON from FilterX values, use `format_json` .

parse_kv

Split a string consisting of whitespace or comma-separated `key=value` pairs (for example, WELF-formatted messages).

Usage: `parse_kv(msg, value_separator="=", pair_separator=", ", stray_words_key="stray_words")`

The `value_separator` must be a single character. The `pair_separator` can consist of multiple characters.

For details, see Parse key=value pairs with FilterX.

parse_leef

Parse a LEEF-formatted string.

Usage: `parse_leef(msg)`

For details, see Parse LEEF with FilterX.

parse_xml

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Parse an XML object into a JSON object.

Usage: `parse_xml(msg)`

For details, see Parse XML with FilterX.

parse_windows_eventlog_xml

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Parses a Windows Event Log XML object into a JSON object.

Usage: `parse_windows_eventlog_xml(msg)`

For details, see Parse Windows Event Log XML with FilterX.

protobuf

Cast a `bytes` or a `protobuf` value into `protobuf` , the type that holds serialized protobuf data. Casting any other type causes an error.

Usage: `protobuf(<bytes or protobuf value>)`

The value isn't validated against a schema. To build a protobuf message from FilterX values using a schema, use `protobuf_message` .

protobuf_message

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Available in AxoSyslog 4.13 and later.

Formats arbitrary data as protobuf using the specified schema.

Usage: `protobuf_data = protobuf_message(my_dict, schema_file="my_schema_file.proto");`

For example, if your proto file looks like this:

```
syntax = "proto2";

message CustomRecord {
  optional string message = 1;
  optional string app = 2;
  optional string host = 3;
  optional int64 pid = 4;
}
```

You can create the following object in FilterX, then include the `protobuf_data` variable in the message:

```
filterx {
  my_dict = {"message":$MESSAGE, "app":$PROGRAM, "host": $HOST, "pid": $PID};
  protobuf_data = protobuf_message(my_dict, schema_file="my_schema_file.proto");
  ${MESSAGE} = {
    "protobuf_data": protobuf_data,
  };
};
```

pubsub_message

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu , or the `axosyslog-grpc` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Available in AxoSyslog 4.10 and later.

Create a Google Pub/Sub message to send with the `google-pubsub-grpc()` destination .

Usage: `pubsub_message(<data>, <attributes>)`

- `data` : The payload of the message. Strings, `bytes` , and `protobuf` values are sent as is, other values are converted to a string.
- `attributes` : A dict of the attributes of the message. The number of attributes can differ from message to message.

Without arguments, `pubsub_message()` returns an empty message. Calling it with one argument, or with an `attributes` argument that isn't a dict, causes an error.

Pass the message to the destination in its `proto-var()` option, instead of `data()` and `attributes()`. For example:

```
log {
  source(s_network);
  filterx {
    $pubsub = pubsub_message(${MESSAGE}, {"host": ${HOST}, "program": ${PROGRAM}});
  };
  destination {
    google-pubsub-grpc(
      project("my-project")
      topic("my-topic")
      proto-var("$pubsub")
    );
  };
};
```

regexp_search

Searches a string and returns the matches of a regular expression as a list or a dictionary. If there are no matches, the result is empty.

> **Note**
>
> - In version 4.9 and earlier, `regexp_search` returned a `dict` or `list` depending on whether named match groups were used in the expression. Starting with version 4.10, `dict` is returned by default. For details, see `list_mode`.
> - Match group zero is now excluded by default unless it's the only match group. To always include the zero match group in the results, use the `keep_zero=true` flag.

Usage: `regexp_search("<string-to-search>", <regular-expression>, <optional-flags=flag_value>)`

For example:

```
# ${MESSAGE} = "ERROR: Sample error message string"
my-variable = regexp_search(${MESSAGE}, "ERROR");
```

You can also use unnamed match groups (`()`) and named match groups (`(?<first>ERROR) (?<second>message)`).

Note the following points:

- Regular expressions are case sensitive by default. For case insensitive matches, add `(?i)` to the beginning of your pattern.
- You can use regexp constants (slash-enclosed regexps) within FilterX blocks to simplify escaping special characters, for example, `/^beginning and end$/` .
- FilterX regular expressions are interpreted in “leave the backslash alone mode” , meaning that a backslash in a string before something that doesn’t need to be escaped and will be interpreted as a literal backslash character. For example, `string\more-string` is equivalent to `string\\more-string` .

Options

You can use the following optional flags in `regexp_search` :

- `keep_zero` : Always return the zero match group. Available in version 4.10 and later. Default value: `false`
- `list_mode` : Return results as a list. Available in version 4.10 and later. Default value: `false`

  If the result is an existing `dict` or `list` object, the function respects the type of the object, even if `list_mode` is set to true.

Unnamed match groups

```
${MY-LIST} = json(); # Creates an empty JSON object
${MY-LIST}.unnamed = regexp_search("first-word second-part third", /(first-word)
(second-part)(third)/);
```

`${MY-LIST}.unnamed` is a list containing: `["first-word second-part third", "first-word", "second-part", "third"],`

Named match groups

```
${MY-LIST} = json(); # Creates an empty JSON object
${MY-LIST}.named = regexp_search("first-word second-part third", /(?<one>first-word)(?<two>second-part)(?<three>third)/);
```

`${MY-LIST}.named` is a dictionary with the names of the match groups as keys, and the corresponding matches as values: `{"0": "first-word second-part third", "one": "first-word", "two": "second-part", "three": "third"},`

Mixed match groups

If you use mixed (some named, some unnamed) groups in your regular expression, the output is a dictionary, where AxoSyslog automatically assigns a key to the unnamed groups. For example:

```
${MY-LIST} = json(); # Creates an empty JSON object
${MY-LIST}.mixed = regexp_search("first-word second-part third", /(?<one>first-word)(second-part)(?<three>third)/);
```

`${MY-LIST}.mixed` is: `{"0": "first-word second-part third", "first": "first-word", "2": "second-part", "three": "third"}`

regexp_subst

Rewrites a string using regular expressions. This function implements the `subst` rewrite rule functionality . If you need only string replacement without regular expression support, use the `str_replace` function as it has better performance.

> **Note**
>
> **Hard macros cannot be modified. For details, see Hard versus soft macros.**

Usage: `regexp_subst(<input-string>, <pattern-to-find>, <replacement>, flags)`

The following example replaces the first `IP` in the text of the message with the `IP-Address` string.

```
regexp_subst(${MESSAGE}, "IP", "IP-Address");
```

To replace every occurrence, use the `global=true` flag:

```
regexp_subst(${MESSAGE}, "IP", "IP-Address", global=true);
```

Note the following points:

- Regular expressions are case sensitive by default. For case insensitive matches, add `(?i)` to the beginning of your pattern.
- You can use regexp constants (slash-enclosed regexps) within FilterX blocks to simplify escaping special characters, for example, `/^beginning and end$/` .
- FilterX regular expressions are interpreted in “leave the backslash alone mode” , meaning that a backslash in a string before something that doesn’t need to be escaped and will be interpreted as a literal backslash character. For example, `string\more-string` is equivalent to `string\\more-string` .

Starting with version 4.10 substitution match groups is enabled by default (use the `groups=false` flag to disable that if needed). You can reference match group indexes up to 999.

```
result = regex_subst("baz,foo,bar", /(\w+),(\w+),(\w+)/, "\\2 \\03 \\1")
```

Options

You can use the following flags with the `regexp_subst` function:

- `global=true` :

  Replace every match of the regular expression, not only the first one.

- `groups=false` :

  Disable substituting match groups. Available in version 4.10 and later.

- `ignorecase=true` :

Do case insensitive match.

- `jit=true` :

  Enable just-in-time compilation function for PCRE regular expressions .

- `newline=true` :

  When configured, it changes the newline definition used in PCRE regular expressions to accept either of the following:

  - a single carriage-return
  - linefeed
  - the sequence carriage-return and linefeed (`\\r` , `\\n` and `\\r\\n` , respectively)

  This newline definition is used when the circumflex and dollar patterns (`^` and `$`) are matched against an input. By default, PCRE interprets the linefeed character as indicating the end of a line. It does not affect the `\\r` , `\\n` or `\\R` characters used in patterns.

- `utf8=true` :

  Use Unicode support for UTF-8 matches: UTF-8 character sequences are handled as single characters.

repr

Available in AxoSyslog 4.11 and later.

Return the string representation of a value, as it would appear in JSON.

Usage: `repr(<value>)`

The difference from `string()` is how strings are handled: `repr()` puts them in double quotes and escapes the special characters, `string()` returns them unchanged. For example, if `${HOST}` is `web01` :

```
string(${HOST});  # web01
repr(${HOST});    # "web01"
```

For other types, both return the same: for example, `repr({"a": 1})` is `{"a":1}` . If a value has no string representation, `repr()` returns an error.

set_fields

Takes a dict and sets multiple fields in it with overrides or defaults (`overrides` and `defaults` are optional parameters).

The `overrides` and `defaults` parameters are also dicts, where:

- the key is the field's name
- the value is either an expression, or a list of expressions.

  If a list is provided, each expression will be evaluated, and the first successful, non-null one is set as the respective field's value. This is similar to chaining null-coalescing (`??`) operators , but has better performance.

`overrides` are always processed for each field. The `defaults` option for a field is only processed if the field isn't set, or it's empty.

For example:

```
set_fields(
  labels,
  overrides={
    "service.name": [service_name, "axosyslog"],
    "service.pid": pid,
  },
  defaults={
    "host.name": [host_name, "localhost"],
  }
);
```

The overrides are equivalent with:

```
labels["service.name"] =?? service_name ?? "axosyslog";
labels["service.pid"] =?? pid;
```

While the defaults section is equivalent with:

```
if (not isset(labels["host.name"])) {
  labels["host.name"] =?? host_name ?? "localhost";
}
```

But using `set_fields` is more readable and has better performance.

set_pri

Available in AxoSyslog 4.11 and later.

Sets the priority value of the message. For example:

```
set_pri(100);
```

The priority value can range from `0` to `191`.

set_timestamp

Available in AxoSyslog 4.11 and later.

Sets the timestamps of the message using a `datetime` value, for example:

```
datetime = strptime("2000-01-01T00:00:00 +0200", "%Y-%m-%dT%H:%M:%S %z");
set_timestamp(datetime, stamp="stamp");
```

The `stamp` argument determines the timestamp to be set: `stamp` or `recvd`. Default value: `stamp`

set_timezone

See set_timezone.

sha1

Available in AxoSyslog 4.25 and later.

Computes the SHA-1 hash of a string or bytes value and returns the result as a lowercase hexadecimal string. To obtain the raw hash as bytes, use the `digest` function with `alg="sha1"` .

Usage: `sha1(string_or_bytes)`

For example:

```
sha1("foobar");  # Returns "8843d7f92416211de9ebb963ff4ce28125932878"
```

sha256

Available in AxoSyslog 4.25 and later.

Computes the SHA-256 hash of a string or bytes value and returns the result as a lowercase hexadecimal string. To obtain the raw hash as bytes, use the `digest` function (`sha256` is also the default algorithm used by `digest` when `alg=` is not set).

Usage: `sha256(string_or_bytes)`

For example:

```
sha256("foobar");  # Returns
"c3ab8ff13720e8ad9047dd39466b3c8974e592c2fa383d4a3960714caef0c4f2"
```

sha512

Available in AxoSyslog 4.25 and later.

Computes the SHA-512 hash of a string or bytes value and returns the result as a lowercase hexadecimal string. To obtain the raw hash as bytes, use the `digest` function with `alg="sha512"` .

Usage: `sha512(string_or_bytes)`

For example:

```
sha512("foobar");
```

startswith

Available in AxoSyslog 4.9 and later.

Returns true if the input string begins with the specified substring. By default, matches are case sensitive. Usage:

```
startswith(input-string, substring);
startswith(input-string, [substring_1, substring_2], ignorecase=true);
```

For details, see String search in FilterX.

strcasecmp

Available in AxoSyslog 4.12 and later.

Usage: `strcasecmp(string1, string2)`

Compare two strings without case sensitivity. Returns:

- `0` if the two strings match,
- a negative value if string1 < string2,
- a positive value if string1 > string2.

For example:

```
strcasecmp("HOST1", "host1"); # Returns 0
```

strftime

Available in AxoSyslog 4.10 and later.

Format datetime values using the specified format string.

Usage: `strftime("format_string", <value-or-variable-to-format>);`

For example:

```
mydate = strptime("2024-04-10T08:09:10Z", "%Y-%m-%dT%H:%M:%S%z");

${MESSAGE} = strftime("%Y-%m-%dT%H-%M-%S %z", my-date);
```

You can use the following format codes in the format string:

- `%a` : The locale's abbreviated weekday name.
- `%A` : The locale's full weekday name.
- `%b` : The locale's abbreviated month name.
- `%B` : The locale's full month name.
- `%c` : The locale's appropriate date and time representation.
- `%C` : The year divided by 100 and truncated to an integer, as a decimal number.
- `%d` : The day of the month as a decimal number [01,31].
- `%D` : Equivalent to `%m / %d / %y` .
- `%e` : The day of the month as a decimal number [1,31]; a single digit is preceded by a space.
- `%f` : Fraction of the second (with or without a leading dot). Width specifies precision, `%6f` means microseconds, `%3f` means milliseconds, `%9f` means nanoseconds. `%f` just means microseconds.
- `%F` : Equivalent to `%+4Y-%m-%d` .
- `%g` : The last 2 digits of the week-based year (see below) as a decimal number [00,99].
- `%G` : The week-based year (see below) as a decimal number (for example, 1977).
- `%h` : Equivalent to %b.
- `%H` : The hour (24-hour clock) as a decimal number [00,23].
- `%I` : The hour (12-hour clock) as a decimal number [01,12].
- `%j` : The day of the year as a decimal number [001,366].
- `%m` : The month as a decimal number [01,12].
- `%M` : The minute as a decimal number [00,59].
- `%n` : A `<newline>` .
- `%p` : The locale's equivalent of either a.m. or p.m.
- `%r` : The time in a.m. and p.m. notation.
- `%R` : The time in 24-hour notation (%H : %M).

- `%S` : The second as a decimal number [00,60].
- `%t` : A `<tab>` .
- `%T` : The time (`%H : %M : %S`).
- `%u` : The weekday as a decimal number [1,7], with 1 representing Monday.
- `%U` : The week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0.
- `%V` : The week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1.
- `%w` : The weekday as a decimal number [0,6], with 0 representing Sunday.
- `%W` : The week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0.
- `%x` : The locale's appropriate date representation.
- `%X` : The locale's appropriate time representation.
- `%y` : The last two digits of the year as a decimal number [00,99].
- `%Y` : The year as a decimal number (for example, 1997).
- `%z` : The offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable
- `%Z` : Same as `%z` , but with the `:` separator (-hh:mm or +hh:mm)

string

Cast a value into a string. Note that currently AxoSyslog evaluates strings and executes template functions and template expressions within the strings. In the future, template evaluation will be moved to a separate FilterX function.

Usage: `string(<string or expression to cast>)`

For example:

```
myvariable = string(${LEVEL_NUM});
```

Sometimes you have to explicitly cast values to strings, for example, when you want to concatenate them into a message using the `+` operator.

str_replace

Available in AxoSyslog 4.15 and later.

Replace a literal string with another one. If you need to use regular expressions, see `regexp_subst` .

Usage: `str_replace(<input-string>, <pattern-to-find>, <replacement>, <max-occurrence>`

If you don't specify the max occurrence, every match is replaced, otherwise only the first `<max-occurrence>` . For example:

```
filterx {
  my_input = "This is an input string";
  my_input = str_replace(my_input, "in", "out");
  # Value of my_input becomes: "This is an output stroutg"

  my_input = "This is an input string";
  my_input = str_replace(my_input, "in", "out", 1);
  # Value of my_input becomes: "This is an output string"
};
```

Note that the search is case sensitive, and supports UTF-8 characters.

str_strip, str_lstrip, str_rstrip

Available in AxoSyslog 4.16 and later.

These functions remove the leading and/or trailing whitespaces from a string, including the `\n\r\t` characters.

strptime

Creates a `datetime` object from a string, similarly to the `date-parser()` . The first argument is the string containing the date. The second argument is a format string that specifies how to parse the date string. Optionally, you can specify additional format strings that are applied in order if the previous one doesn't match the date string.

Usage: `strptime(time_str, format_str_1, ..., format_str_N)`

For example:

```
${MESSAGE} = strptime("2024-04-10T08:09:10Z", "%Y-%m-%dT%H:%M:%S%z");
```

> **Note**
>
> **If none of the format strings match, `strptime` returns the null value and logs an error message to the `internal()` source of AxoSyslog. If you want the FilterX block to explicitly return false in such cases, use the `isset` FilterX function on the result of `strptime`.**

You can use the following format codes in the format string:

- `%a`: The day of week, using the locale's weekday names. Either the abbreviated or full name may be specified.
- `%A`: Same as % `a`.
- `%b`: The month, using the locale's month names. Either the abbreviated or full name may be specified.
- `%B`: The same as `%b`.
- `%c`: The date and time, using the locale's date and time format.
- `%C`: The century number [0,99]. Leading zeros are permitted but not required. This conversion should be used in conjunction with the %y conversion.
- `%d`: The day of month [1,31]. Leading zeros are permitted but not required.
- `%D`: The date as `%m/%d/%y`.
- `%e`: The same as `%d`.
- `%F`: The date as `%Y-%m-%d` (the ISO 8601 date format).
- `%g`: The year corresponding to the ISO week number, without the century. (A NetBSD extension.)
- `%G`: The year corresponding to the ISO week number, with the century. (A NetBSD extension.)
- `%h`: The same as `%b`.
- `%H`: The hour (24-hour clock) [0,23]. Leading zeros are permitted but not required.
- `%I`: The hour (12-hour clock) [1,12]. Leading zeros are permitted but not required.
- `%j`: The day number of the year [1,366]. Leading zeros are permitted but not required.
- `%k`: The same as `%H`.
- `%l`: The same as `%I`.
- `%m`: The month number [1,12]. Leading zeros are permitted but not required.

- `%M` : The minute [0,59]. Leading zeros are permitted but not required.
- `%n` : Any white-space, including none.
- `%p` : The locale's equivalent of a.m. or p.m.
- `%r` : The time (12-hour clock) with `%p` , using the locale's time format.
- `%R` : The time as `%H:%M` .
- `%S` : The seconds [0,60]. Leading zeros are permitted but not required.
- `%s` : The number of seconds since the Epoch, UTC (see mktime(3)). (A NetBSD extension.)
- `%f` : Fraction of the second (with or without a leading dot).
- `%t` : Any white-space, including none.
- `%T` : The time as `%H:%M:%S` .
- `%u` : The day of the week as a decimal number, where Monday = 1. (A NetBSD extension.)
- `%U` : The week number of the year (Sunday as the first day of the week) as a decimal number [0,53]. Leading zeros are permitted but not required. All days in a year preceding the first Sunday are considered to be in week 0.
- `%V` : The ISO 8601:1988 week number as a decimal number. If the week (starting on Monday) that contains January 1 has more than three days in the new year, then it is considered the first week of the year. If it has fewer than four days in the new year, then it is considered the last week of the previous year. Weeks are numbered from 1 to 53. (A NetBSD extension.)
- `%w` : The weekday as a decimal number [0,6], with 0 representing Sunday. Leading zeros are permitted but not required.
- `%W` : The week number of the year (Monday as the first day of the week) as a decimal number [0,53]. Leading zeros are permitted but not required. All days in a year preceding the first Monday are considered to be in week 0.
- `%x` : The date, using the locale's date format.
- `%X` : The time, using the locale's time format.
- `%y` : The year within the 20th century [69,99] or the 21st century [0,68]. Leading zeros are permitted but not required. If specified in conjunction with `%C` , specifies the year [0,99] within that century.
- `%Y` : The year, including the century (i.e., 1996).
- `%Z` : Timezone in ascii format (for example, PST), or in format -/+0000, accepts `:` in the middle of timezones (ISO 8601)
- `%z` : Timezone in ascii format (for example, PST), or in format -/+0000, accepts `:` in the middle of timezones (ISO 8601) (Required element)
- `%%` : matches a literal `%` . No argument is converted.

> **Warning**
>
> When using the `%z` and `%Z` format codes, consider that while `%z` strictly expects a specified timezone, and triggers a warning if the timezone is missing, `%Z` does not trigger a warning if the timezone is not specified.
>
> For further information about the `%z` and `%Z` format codes, see the 'DESCRIPTION' section on the srtptime(3) - NetBSD Manual Pages .

For example, for the date `01/Jan/2016:13:05:05 PST` use the following format string: `"%d/%b/%Y:%H:%M:%S %Z"`

The `isodate` FilterX function is a specialized variant of `strptime`, that accepts only a fixed format.

subnet

See subnet.

tuple

See Tuples.

unset

Deletes a variable, a name-value pair, or a key in a complex object (like JSON), for example: `unset(${<name-value-pair-to-unset>});`

> **Note**
>
> **You cannot unset hard macros .**

You can also list multiple values to delete: `unset(${<first-name-value-pair-to-unset>}, ${<second-name-value-pair-to-unset>});`

See also Delete values.

unset_empties

Deletes (unsets) the empty fields of an object, for example, a JSON object or list. By default, the object is processed recursively, so the empty values are deleted from inner dicts and lists as well. If you set the `replacement` option, you can also use this function to replace fields of the object to custom values.

Usage: `unset_empties(object, options)`

The `unset_empties()` function has the following options:

- `ignorecase` : Set to `true` to perform case-insensitive matching. Default value: `false` . Available in AxoSyslog 4.9 and later, default changed to `false` in version 4.10.
- `recursive` : Enables recursive processing of nested dictionaries. Default value: `true`
- `replacement` : Replace the target elements with the value of `replacement` instead of removing them. Available in AxoSyslog 4.9 and later.
- `targets` : A list of elements to remove or replace. Default value: `["", null, [], {}]` . Available in AxoSyslog 4.9 and later.

For example, to remove the fields with `-` and `N/A` values, you can use

```
unset_empties(input_object, targets=["-", "N/A"], ignorecase=false);
```

update_metric

Updates a labeled metric counter, similarly to the `metrics-probe()` parser .

Usage: `update_metric("key", labels={"key": "value"}, increment=1|set=20, level=0)`

The `set` option is available in AxoSyslog 4.28 and later. For details, see Create metrics with FilterX.

upper

Converts all characters of a string uppercase characters.

Usage: `upper(string)`

urldecode

Available in AxoSyslog 4.25 and later.

Decodes a percent-encoded URL string and returns the decoded value as a string. An invalid percent-encoding causes a runtime error.

Usage: `urldecode(string)`

For example:

```
urldecode("example%2Furl%3Fwith%3D1%26parameters%3D2");
# Returns "example/url?with=1&parameters=2"
```

urlencode

Available in AxoSyslog 4.25 and later.

Percent-encodes a string so that it's safe to include in a URL. Characters outside the unreserved set are replaced with their `%HH` escape.

Usage: `urlencode(string)`

For example:

```
urlencode("example/url?with=1&parameters=2");
# Returns "example%2Furl%3Fwith%3D1%26parameters%3D2"
```

utf8_sanitize

Available in AxoSyslog 4.25 and later.

Returns a string where invalid UTF-8 byte sequences in the input are replaced with their `\xNN` escaped representation. If the input is already valid UTF-8, the original string is returned unchanged. The function is idempotent: calling it on an already sanitized value produces the same result.

Usage: `utf8_sanitize(string)`

For example:

```
utf8_sanitize("example");         # Returns "example"
utf8_sanitize("\x80\x81\x82");    # Returns "\x80\x81\x82" (as literal characters)
```

utf8_validate

Available in AxoSyslog 4.25 and later.

Returns `true` if the input string contains only valid UTF-8 byte sequences, and `false` otherwise.

Usage: `utf8_validate(string)`

For example:

```
utf8_validate("example");       # Returns true
utf8_validate("\x80\x81\x82");  # Returns false
```

uuid or uuid4

Available in AxoSyslog 4.25 and later. The `uuid4` alias is available in version 4.27 and later.

Generates a random RFC 9562 UUIDv4 identifier. Usage:

```
uuid()
```

For example:

```
${MESSAGE} = string(uuid());
```

uuid7

Available in AxoSyslog 4.27 and later.

Generates a random RFC 9562 UUIDv7 identifier, which embeds a millisecond-precision Unix timestamp, so they sort lexically by creation time.

Usage:

```
uuid7()
```

For example:

```
${MESSAGE} = string(uuid7());
```

vars

Returns the variables (including pipeline variables and name-value pairs) defined in the FilterX block as a JSON object. The names of name-value pairs begins with the `$` character. To exclude name-value pairs, set the `exclude_msg_values=true` flag.

For example:

```
filterx {
  ${logmsg_variable} = "foo";
  local_variable = "bar";
  declare pipeline_level_variable = "baz";
  ${MESSAGE} = vars();
};
```

The value of `${MESSAGE}` will be: `{"$logmsg_variable":"foo","pipeline_level_variable":"baz"}`

10.19 Troubleshoot FilterX

Fix common FilterX problems, like dropped messages and missing fields, and track failing statements with the failure_info functions.

Common problems

Messages are dropped unexpectedly

AxoSyslog drops the message from the log path if any statement of the FilterX block is falsy, or results in an error. For example, referencing a field or variable that doesn't exist is an error. For details, see Truthy and falsy values .

To handle fields that exist only in some messages:

- Set a default value with the null coalescing operator (`??`) .
- Put the statements that use the field into a conditional statement .

To find out which statement failed, see Track failures .

Fields are missing from the output

Only macros and name-value pairs (names starting with `$`) reach the destination. Local and pipeline variables don't, and template functions can't access them either. Assign the value to a name-value pair to send it. For details, see FilterX variables in destinations .

A comparison doesn't match

How FilterX compares two values depends on their types: two strings are compared as strings, and case sensitively. If one side is a number, the comparison is numeric. For details, see String and numerical comparison .

An IP address check is always false

The address family of the IP address and the subnet must match. Checking an IPv4 address against an IPv6 subnet (or the other way around) is falsy, so AxoSyslog drops the message. For details, see Check whether an IP address is in a subnet .

A parser returns an error or null

If the input doesn't match the expected format, the parser function fails, and the message is dropped. For example:

- `json()` returns an error if the string isn't valid JSON. See Parse JSON with FilterX.
- `parse_windows_eventlog_xml()` returns false if the input isn't valid XML, or doesn't use the Windows Event Log schema. See Parse Windows Event Log XML with FilterX.
- `strptime()` returns null if none of the format strings match. See Parse dates and timestamps with FilterX.

Changes to OpenTelemetry messages are lost

If you modify messages received with the `opentelemetry()` source, you must write the modified objects back to the raw data structures, otherwise the destination sends the original message. For details, see Modify incoming OTEL .

Track failures

To help troubleshooting FilterX blocks, AxoSyslog includes some specific functions that allow you to track failures in FilterX code:

- `failure_info_enable()`: Collect failure information from this point downwards through all branches of the pipeline. By default, only truthy expressions are collected. To collect collect falsy evaluations as well, use `failure_info_enable(collect_falsy=true);`
- `failure_info_clear()`: Clear all failure information collected so far.
- `failure_info_meta({})`: Attach metadata to the given section of FilterX code. The metadata remains in effect until the next call, or until the end of the enclosing FilterX block, whichever comes first. For example, you can use this function to mark where you are in a decision tree:

  ```
  failure_info_meta({"step": "#1 step-description"});
  ```

- `failure_info()`: Return the collected failure information as a FilterX dictionary. Call this function as late as possibly, for example, in the last log path of your AxoSyslog configuration, or within a fallback path. The output looks like:

```
[
  {
    "meta": {
      "step": "Setting common fields"
    },
    "location": "/etc/syslog-ng/syslog-ng.conf:33:7",
    "line": "nonexisting.key = 13;",
    "error": "No such variable: nonexisting"
  }
]
```

The following is an example configuration that uses these functions:

```
destination console_output {
  stdout(template("$a\n"));
};

source input {
  channel {
    source { stdin(); network(port(4444)); };

    filterx {
      # it can be enabled for a subset of messages
      failure_info_enable(collect_falsy=true);
    };
  };
};

log {
  source(input);

  log "log-path-1" {
    filterx {
      # "log-path-1" was successful, clear accumulated errors
      failure_info_clear();
    };
  };

  log "log-path-2" {
```

```
    filterx {

      # Step #1: abc
      failure_info_meta({"step": "#1 log-path-2"});
      a = 1;
      b = 3;
      1 == 1;
      true;

      # Step #2: cba
      failure_info_meta({"step": "#2 log-path-2"});
      declare g = 33;
      nonexisting.key = g;
    };
  };

  log "log-path-3" {
    filterx {
      failure_info_meta({"step": "falsystep"});
      1 == 0;
      true;
    };
  };
};

# WARNING: use the last logpath in the config file, or a real fallback path
log "fallback" {
  source(input);

  filterx {
    $a = failure_info();
  };

  destination(console_output);
};
```

If you start AxoSyslog with this configuration, the output will look like this (because it's trying to assign a value to a non-existing variable):

```
[
    {
    "meta": {
        "step": "Setting common fields"
    },
    "location": "/etc/syslog-ng/syslog-ng.conf:33:7",
    "line": "nonexisting.key = 13;",
    "error": "No such variable: nonexisting"
    }
]
```

11 Global options

Configure the options that apply to the whole AxoSyslog configuration, and look up every global option in the reference.

11.1 Configuring global 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 options statement to the `syslog-ng.conf` 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.conf` configuration file:

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

For a detailed list of the available options, see Global options reference. For important global options and recommendations on their use, see Best practices and examples.

11.2 Global options reference

The following options can be specified in the options statement, as described in Configuring global 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 AxoSyslog 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 AxoSyslog 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

chain-hostnames() setting on the server	no	client-hostname-resolved-on-the-server	

If the log message is forwarded to the AxoSyslog server via a AxoSyslog relay, the hostname depends on the settings of the `keep-hostname()` and the `chain-hostnames()` options both on the AxoSyslog relay and the AxoSyslog 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	

chain-hostnames() setting 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	relay-hostname-resolved-on-the-server

chain-hostnames() setting on the relay	no	keep-hostname() setting on the relay	no	client-hostname-resolved-on-the-relay	client-hostname-resolved-on-the-relay / relay-hostname-resolved-on-the-ser-

	ver	client-hostname-resolved-on-the-server

The `chain-hostnames()` option can interfere with the way AxoSyslog counts the log source hosts. As a result, AxoSyslog falsely perceives several 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:

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)

- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

The `check-hostname()` global option applies to the following sources: `file()` , `network()` , `pipe()` , `program()` , `stdin()` , `syslog()` , `systemd-syslog()` , `unix-dgram()` , `unix-stream()` , `wildcard-file()` . Instead of using the global option, you can also set the `check-hostname()` option for the specific source.

For the `python()` and `python-fetcher()` sources and the `syslog-parser()` parser you can enable this option as a flag.

check-program()

Accepted values:	`yes` , `no`
Default:	`no`

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior per source using the `check-program` source flag.

create-dirs()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Enable or disable directory creation for destination files and sockets.

custom-domain()

> **Note**
>
> **This global option works only if the `use-fqdn()` global option is set to `yes`.**

Accepted values:	string
Default:	`empty string`

Description: Use this option to specify a custom domain name that is appended after the short hostname to receive the fully qualified domain name (FQDN). This option affects every outgoing message: eventlog sources, file sources, MARK messages and internal messages of AxoSyslog.

- If the hostname is a short hostname, the custom domain name is appended after the hostname (for example, `mypc` becomes `mypc.customcompany.local`).
- If the hostname is an FQDN, the domain name part is replaced with the custom domain name (for example, if the FQDN in the forwarded message is `mypc.mycompany.local` and the custom domain name is `customcompany.local`, the hostname in the outgoing message becomes `mypc.customcompany.local`).

dir-group()

Accepted values:	groupid
Default:	root

Description: The default group for newly created directories.

dir-owner()

Accepted values:	userid
Default:	root

Description: The default owner of newly created directories.

dir-perm()

Accepted values:	permission value
Default:	-1

Description: The permission mask of directories created by `syslog-ng`. Log directories are only created if a file after macro expansion refers to a non-existing directory, and directory creation is enabled (see also the `create-dirs()` option). For octal numbers prefix the number with `0`, for example, use `0755` for `rwxr-xr-x`.

To preserve the original properties of an existing directory, use the option without specifying an attribute: `dir-perm()`. Note that when creating a new directory without specifying attributes for `dir-perm()`, the default permission of the directories is masked with the umask of the parent process (typically `0022`).

Starting with version 3.16, the default value of this option is -1, so AxoSyslog does not change the ownership, unless explicitly configured to do so.

disk-buffer()

Description: Sets default values for the disk-buffer sub-options listed below. Each destination that uses `disk-buffer()` inherits these defaults unless the destination overrides them. Other `disk-buffer()` sub-options (for example, `capacity-bytes()`, `dir()`, `reliable()`) must be set on the destination itself – see Using disk-based and memory buffering.

```
options {
    disk-buffer(
        prealloc(no)
        truncate-size-ratio(0.1)
        stats(freq(60))
    );
};
```

prealloc()

Type:	**yes/no**
Default:	no

Available in AxoSyslog 4.0 and later.

Description:

By default, AxoSyslog doesn't reserve the disk space for the disk-buffer file, since in a properly configured and sized environment the disk-buffer is practically empty, so a large preallocated disk-buffer file is just a waste of disk space. But a preallocated buffer can prevent other data from using the intended buffer space (and elicit a warning from the OS if disk space is low), preventing message loss if the buffer is actually needed. To avoid this problem, when using AxoSyslog 4.0 or later, you can preallocate the space for your disk-buffer files by setting `prealloc(yes)`.

In addition to making sure that the required disk space is available when needed, preallocated disk-buffer files provide radically better (3-4x) performance as well: in case of an outage the amount of messages stored in the disk-buffer is continuously growing, and using large continuous files is faster, than constantly waiting on a file to change its size.

If you are running AxoSyslog on a dedicated host (always recommended for any high-volume settings), use `prealloc(yes)`.

stats(freq())

Accepted values:	number (seconds)
Default:	0 (disabled)

Description: The frequency (in seconds) at which AxoSyslog emits disk-buffer metrics to the internal statistics. `0` disables periodic reporting. Disk-buffer metrics remain available through `syslog-ng-ctl stats` regardless of this setting.

truncate-size-ratio()

Type:	**number (between 0 and 1)**
Default:	1 (do not truncate)

Description: Limits the truncation of disk-buffer files. Truncation can slow down disk I/O operations but saves disk space. AxoSyslog only truncates a disk-buffer file if the possible disk gain is more than `truncate-size-ratio()` times `capacity-bytes()`. For details and trade-offs, see `truncate-size-ratio()` on the destination-level disk-buffer page.

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); };
```

filterx-jit()

Accepted values:	`yes` , `no`
Default:	`no`

Available in AxoSyslog version 4.26 and later.

Description: Enables just-in-time (JIT) compilation of the FilterX blocks of the configuration. JIT compilation can improve the throughput of configurations that do heavy FilterX processing, at the cost of extra work when AxoSyslog starts or reloads. For details on FilterX, see FilterX.

```
options { filterx-jit(yes); };
```

> **Note**
>
> **This option requires a build of AxoSyslog that includes FilterX JIT support. If you set `filterx-jit(yes)` on a build that does not have it, AxoSyslog logs an error and does not start.**

filterx-jit-debug-info()

Accepted values:	`filterx` , `llvm-ir`
Default:	`filterx`

Available in AxoSyslog version 4.26 and later.

Description: Sets the kind of debug information that the FilterX JIT compiler generates. This option only has an effect if you also enable `filterx-jit()` .

- `filterx` : Generate debug information that maps the compiled code back to your FilterX source. This is the default.
- `llvm-ir` : Generate debug information that maps the compiled code to the underlying LLVM intermediate representation. Use this when you are debugging the JIT compiler itself. This mode is not available on every platform: if the build does not support it, AxoSyslog logs an error and rejects the configuration.

flush-lines()

Accepted values:	number
Default:	100

Description: Specifies how many lines are flushed to a destination at a time. The AxoSyslog 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 AxoSyslog application flushes the messages if it has sent `flush-lines()` number of messages, or the queue became empty. If you stop or reload AxoSyslog or in case of network sources, the connection with the client is closed, AxoSyslog automatically sends the unsent messages to the destination.

UDP and other datagram transports write one message per packet.

flush-timeout() (DEPRECATED)

Type:	time in milliseconds
Default:	10000 [milliseconds]

This option is deprecated. Several destination drivers support the `batch-timeout()` option instead, which you can set at the destination level.

Description: Specifies the time AxoSyslog waits for lines to accumulate in the output buffer. The AxoSyslog application sends flushes to the destinations evenly. The timer starts when the first message arrives to the buffer, so if only few messages arrive, AxoSyslog sends messages to the destination at most once every `flush-timeout()` seconds.

You can also set this option per destination.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

group()

Accepted values:	groupid
Default:	root

Description: The default group of output files. By default, `syslog-ng` changes the privileges of accessed files (for example, `/dev/null`) to `root.root 0600`. To disable modifying privileges, use this option with the `-1` value.

jvm-options()

Type:	list
Default:	N/A

Description: Specify the Java Virtual Machine (JVM) settings of your Java destination from the AxoSyslog 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)`), AxoSyslog assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep-hostname(no)`), AxoSyslog 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 AxoSyslog. For details on using name resolution in AxoSyslog, see Using name resolution in syslog-ng.

> **Note**
>
> If the log message does not contain a hostname in its `HOST` field, AxoSyslog 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, AxoSyslog 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 AxoSyslog server and also on every relay, otherwise AxoSyslog will treat incoming messages as if they were sent by the last relay.**

keep-timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether AxoSyslog 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.

> **Warning**
>
> **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

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.

You can also set this option per destination.

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-fifo-size()

Accepted values:	number (messages)
Default:	10000

Description: The number of messages that the output queue can store.

log-flow-control()

Accepted values:	`yes`, `no`
Default:	`no`

Available in AxoSyslog 4.12 and later.

Description: Enables flow control for all log paths. When set to yes, flow control is globally enabled, but you can selectively disable it for individual log paths using the `no-flow-control` flag. For example:

```
options {
  log-flow-control(yes);
};

log {
  source { system(); };
  destination { network("server" port(5555)); };
  flags(no-flow-control);
};

log { ... };
```

> **CAUTION:**
>
> Enabling global flow control can cause the `system()` source to block. As a result, if messages accumulate at the destination, applications that log through the system may become completely stalled, potentially halting their operation. We don't recommend enabling flow control in log paths that include the `system()` source.

log-iw-size()

Type:	number
Default:	100

Description: Specifies the source window size - the maximum number of in-flight messages permitted by the source before flow control is enforced. This only applies when `flow-control` is enabled.

> **CAUTION:**
>
> If you change the value of `log-iw-size()` and `keep-alive()` is enabled, the change will affect only new connections, the `log-iw-size()` of kept-alive connections will not change. To apply the new `log-iw-size()` value to every connection, restart the `syslog-ng` service . A simple configuration reload is *NOT* sufficient.

If the source is receiving data using the UDP protocol, always restart the `syslog-ng` service after changing the value of `log-iw-size()` for the changes to take effect.

Note that when using `disk-buffer()`, the messages stored on disk are not included in the window size calculation. For details about the effects of this parameter, see Managing incoming and outgoing messages with flow-control.

You can also set this option per source.

log-level()

Accepted values:	`default`, `verbose`, `debug`, `trace`
Default:	`default`

Description: Controls AxoSyslog's own internal log level. Corresponds to setting the internal log level using `syslog-ng-ctl` or the command line options of `syslog-ng` (the `-d`, `-v`, and `-t`). Setting the log level in the configuration makes it easier to control logging in containerized environments where changing command line options is more problematic.

Available in AxoSyslog 4.0 and later.

Higher log-levels automatically include messages from lower log-levels:

- `default`: Just normal log messages.
- `verbose`: Normal and verbose log messages.
- `debug`: Include debug messages of AxoSyslog.
- `trace`: Include trace messages of how messages are processed.

```
options {
  log-level(debug);
};
```

log-msg-size()

Accepted values:	number (bytes)
Default:	65536

Description: Maximum length of an incoming 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 (256 MiB).

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

> **Note**
>
> **In most cases, `log-msg-size()` does not need to be set higher than 10 MiB.**

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

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

long-hostnames()

Obsolete alias for `chain-hostnames()` .

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 AxoSyslog version 1.6.x.

mark-freq()

Accepted values:	number [seconds]
Default:	1200

Description: An alias for the obsolete `mark()` option, retained for compatibility with 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 AxoSyslog 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 AxoSyslog.

> **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 AxoSyslog 3.4 and later.

min-iw-size-per-reader()

Accepted values:	number
Default:	100

Description: Sets the minimum initial window size allocated to each reader when multiple readers (for example, in a `wildcard-file()` source) share the same `log-iw-size()` budget. The effective per-reader window is `max(log-iw-size() / number-of-readers, min-iw-size-per-reader())`. For details on flow control, see Managing incoming and outgoing messages with flow-control.

normalize-hostnames()

Accepted values:	`yes`, `no`
Default:	`no`

Description: If enabled (`normalize-hostnames(yes)`), AxoSyslog 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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `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 AxoSyslog does not change the ownership, unless explicitly configured to do so.

pass-unix-credentials() (DEPRECATED)

Accepted values:	`yes` or `no`
Default:	yes

Deprecated since AxoSyslog 3.35. Use the `so-passcred()` source option to control credential passing per source, or the `ignore-aux-data` source flag to skip collecting the credentials altogether.

Description: Enable AxoSyslog 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 AxoSyslog 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 AxoSyslog 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()

Available in AxoSyslog 4.1 and later.

Description: The `stats()` option is a collection of statistics-related options.

```
options {
    stats(
        freq(1)
        level(1)
        lifetime(1000)
        max-dynamics(10000)
        syslog-stats(yes)
    );
};
```

freq()

Accepted values:	number
Default:	0 (seconds)

Description: The period between two STATS messages in seconds. STATS are log messages sent by `syslog-ng`, containing statistics about dropped log messages. `0` disables the STATS messages.

> **Note**
>
> Starting with AxoSyslog version 4.12, the default value of this option is `0`, disabling STATS messages. We recommend accessing metrics using the `syslog-ng-ctl stats` interface for monitoring and observability, for example, by using `syslog-ng-ctl stats prometheus`.
>
> In earlier versions, the default was `600` (ten minutes).

healthcheck-freq()

Accepted values:	number (seconds)
Default:	300 (5 minutes)

Description: Controls how frequently AxoSyslog publishes health-check results as metrics. Health checks verify that the internal processing pipeline responds in a timely manner. For the list of emitted metrics and how to inspect them, see `syslog-ng-ctl`.

level()

Accepted values:	`0`, `1`, `2`, `3`
Default:	`0`

Description: Specifies the detail of metrics and statistics AxoSyslog 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, see Statistics and metrics of AxoSyslog.

lifetime()

Accepted values:	number (seconds)

Default:	`N/A`

Description: Dynamic counters in metrics are pruned after `lifetime` expires. Note that orphaned counters are not pruned (you can prune them by running `syslog-ng-ctl stats --remove-orphans`).

max-dynamics()

Accepted values:	number
Default:	`N/A`

Available in AxoSyslog 4.1 and later.

Description: To avoid performance issues or even overloading AxoSyslog (for example, if a script starts to send logs from different IP addresses to AxoSyslog), you might want to limit the number of registered dynamic counters in the message statistics. For details on message statistics, see Statistics and metrics of AxoSyslog.

- 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.

  > **Warning**
  >
  > **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 don't use statistics, or if you aren't 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 hasn't been configured before) and restart AxoSyslog, 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.**

syslog-stats()

Accepted values:	`yes`, `no`, `auto`
Default:	`auto`

Available in AxoSyslog 4.1 and later.

Description: Changes the behavior of counting messages based on different syslog fields, like `SEVERITY`, `FACILITY`, `HOST`.

Possible values:

- `yes`: Enable syslog stats.
- `no`: Disable syslog stats.
- `auto`: Use the setting of the old `stats(level())` option.

stats-freq()

Deprecated legacy option. Use `stats(freq())` instead.

stats-level()

Deprecated legacy option. Use `stats(level())` instead.

stats-lifetime()

Deprecated legacy option. Use `stats(lifetime())` instead.

stats-max-dynamics()

Deprecated legacy option. Use `stats(max-dynamics())` instead.

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), AxoSyslog 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 AxoSyslog waits for identical messages.

You can also set this option per destination.

sync() or sync-freq() (DEPRECATED)

Accepted values:	number (messages)
Default:	0

Description: Obsolete aliases for `flush-lines()`

template-escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the `'`, `"`, and backslash 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.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

You can also set this option per destination.

threaded()

Accepted values:	`yes` or `no`
Default:	yes

Description: Enable AxoSyslog to run in multithreaded mode and use multiple CPUs. Available only in AxoSyslog 3.3 and later. Note that setting `threaded(no)` does not mean that AxoSyslog will use only a single thread. For details, see Multithreading and scaling.

time-reap()

Accepted values:	number (seconds)
Default:	60 or 0, see description for details

Description: The time to wait in seconds before an idle destination file or pipe is closed. Note that only destination files having macros in their filenames are closed automatically.

Starting with version 3.23, the way how `time-reap()` works is the following.

1. If the `time-reap()` option of the destination is set, that value is used, for example:

```
destination d_fifo {
    pipe(
        "/tmp/test.fifo",
        time-reap(30)  # sets time-reap() for this destination only
    );
};
```

2. If the `time-reap()` option of the destination is not set, and the destination does not use a template or macro in its filename or path, `time-reap()` is automatically set to 0. For example:

```
destination d_fifo {
    pipe(
        "/tmp/test.fifo",
    );
};
```

3. Otherwise, the value of the global `time-reap()` option is used, which defaults to 60 seconds.

time-reopen()

Accepted values:	number [seconds]
Default:	60

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

time-sleep() (DEPRECATED)

Accepted values:	number
Default:	0

Description: The time to wait in milliseconds between each invocation of the `poll()` iteration.

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.

trim-large-messages()

Accepted values:	`yes` or `no`
Default:	no

Description: Determines what AxoSyslog does with incoming log messages that are received using the IETF-syslog protocol using the `syslog()` driver, and are longer than the value of `log-msg-size()`. Other drivers ignore the `trim-large-messages()` option.

- If set to `no`, AxoSyslog drops the incoming log message.
- If set to `yes`, AxoSyslog trims the incoming log message to the size set in `log-msg-size()`, and adds the `trimmed` tag to the message. The rest of the message is dropped. You can use the tag to filter on such messages.

```
filter f_trimmed {
    tags("trimmed");
};
```

If AxoSyslog trims a log message, it sends a debug-level log message to its `internal()` source.

As a result of trimming, a parser could fail to parse the trimmed message. For example, a trimmed JSON or XML message will not be valid JSON or XML.

Available in AxoSyslog version 3.21 and later.

ts-format()

Accepted values:	`rfc3164` , `bsd` , `rfc3339` , `iso`
Default:	`rfc3164`

Description: Specifies the timestamp format used when AxoSyslog 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.

> **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 a file (for example, from `/etc/hosts`) set using the `dns-cache-hosts()` global option . The AxoSyslog application blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your AxoSyslog network

endpoint with firewall rules, and make sure that all hosts which may get to AxoSyslog 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.

For details on configuring local DNS resolution, see Using name resolution in syslog-ng.

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

use-fqdn()

Type:	yes or no
Default:	no

Description: Use this option to add a Fully Qualified Domain Name (FQDN) instead of a short hostname. You can specify this option either globally or per-source. The local setting of the source overrides the global option if available.

> **Note**
>
> **Set `use-fqdn()` to `yes` if you want to use the `custom-domain()` global option.**

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

use-rcptid() (DEPRECATED)

Accepted values:	`yes` or `no`
Default:	`no`

This option is deprecated, use the `use-uniqid()` option instead.

Description: When the `use-rcptid` global option is set to `yes`, AxoSyslog 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 64-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 AxoSyslog 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

12 TLS-encrypted message transfer

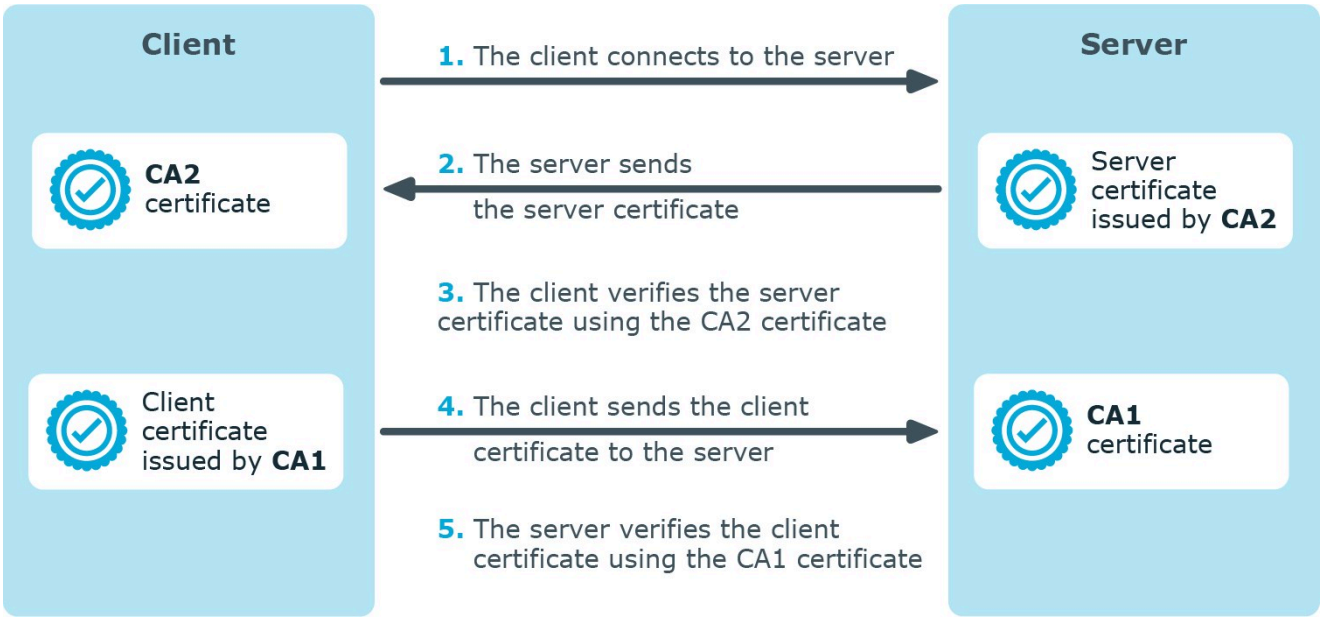
Encrypt log transport with TLS, set up server or mutual authentication, use password-protected keys, and look up the TLS options.

12.1 Secure logging using TLS

The AxoSyslog 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 AxoSyslog nodes, or to send log data to a 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:



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 AxoSyslog server that identifies the server.
- The certificate of the Certificate Authority that issued the certificate of the AxoSyslog server (or the self-signed certificate of the AxoSyslog server) must be available on the AxoSyslog client.

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

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

Mutual authentication ensures that the AxoSyslog server accepts log messages only from authorized clients.

For more information about configuring TLS communication, see Encrypting log messages with TLS.

For more information about TLS-related error messages, see Error messages.

12.2 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.

12.2.1 Configuring TLS server authentication on the AxoSyslog clients

Purpose:

Complete the following steps on every AxoSyslog 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 AxoSyslog server (or the self-signed certificate of the AxoSyslog server) to the AxoSyslog 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.conf` 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 AxoSyslog 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 AxoSyslog 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.

> **Warning**
>
> The encrypted connection between the server and the client fails if the `Common Name` or the `subject_alt_name` parameter of the server certificate does not contain the hostname or the IP address (as resolved from the AxoSyslog clients and relays) of the server.
>
> Do not forget to update the certificate files when they expire.

12.2.2 Configuring TLS server authentication on the AxoSyslog server

Purpose:

Complete the following steps on the AxoSyslog server:

Steps:

1. Create an X.509 certificate for the AxoSyslog 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 AxoSyslog server to the AxoSyslog 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 AxoSyslog server to the AxoSyslog 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.conf` 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog server. The identity of the AxoSyslog 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)
        )
    );
};
```

Warning

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

12.3 Mutual authentication using TLS

This section describes how to configure mutual authentication between the AxoSyslog 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.

12.3.1 Configuring mutual TLS authentication on the AxoSyslog clients

Purpose:

Complete the following steps on every AxoSyslog 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 AxoSyslog client.

2. Copy the certificate (for example, `client_cert.pem`) and the matching private key (for example, `client.key`) to the AxoSyslog 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 AxoSyslog server (or the self-signed certificate of the syslog-ng server) to the AxoSyslog 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.conf` 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 AxoSyslog 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 AxoSyslog 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.

Warning

The encrypted connection between the server and the client fails if the `Common Name` or the `subject_alt_name` parameter of the server certificate does not match the hostname or the IP address (as resolved from the AxoSyslog clients and relays) of the server.

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

12.3.2 Configuring mutual TLS authentication on the AxoSyslog server

Purpose:

Complete the following steps on the AxoSyslog server:

Steps:

1. Copy the certificate (for example, `syslog-ng.cert`) of the AxoSyslog server to the AxoSyslog 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 AxoSyslog clients to the AxoSyslog 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 AxoSyslog server to the AxoSyslog 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.conf` 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 AxoSyslog 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")
        )
    );
};
```

Warning

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

Note: identifiers of the certificate presented by the clients will be available under the `.tls.x509_*` macros. See .tls.x509.

12.4 Password-protected keys

Starting with AxoSyslog version 3.14, you can use password-protected private keys in the `network()` and `syslog()` source and destination drivers.

Restrictions and limitations

> **Note**
>
> *Hazard of data loss!* If you use password-protected keys, you must provide the passphrase of the password-protected keys every time AxoSyslog is restarted (AxoSyslog 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 AxoSyslog configuration will be unaffected.
>
> This means that if you use a password-protected key in a destination, and you use this destination in a log path that has multiple destinations, neither destinations will receive log messages until you provide the password. In this cases, always use 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 AxoSyslog can use at the same time might be limited (for example, on Ubuntu 16.04 you can store 16 passwords if you are running AxoSyslog as a non-root user). If you use lots of password-protected private keys in your AxoSyslog 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 AxoSyslog 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 AxoSyslog cannot use it. The sources and destinations that use these keys will not work until you provide the passwords. Other parts of the AxoSyslog configuration will be unaffected. You must provide the passphrase of the password-protected keys every time AxoSyslog 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. AxoSyslog 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-ctl manual page.

12.5 TLS options

This page describes the TLS-related options of the `network()` and `syslog()` drivers. Where applicable, other drivers also support encrypted transport, see the documentation of the other drivers for details.

> **Note**
>
> **The format of the TLS connections used by AxoSyslog is similar to using AxoSyslog and stunnel, but the source IP information is not lost.**

To encrypt connections, use the `transport("tls")` and `tls()` options in the source and destination statements.

The tls() option can include the following settings:

allow-compress()

Accepted values:	`yes` or `no`
Default:	`no`

Description: Enable on-the-wire compression in TLS communication. Note that this option must be enabled both on the server and the client to have any effect. Enabling compression can significantly reduce the bandwidth required to transport the messages, but can slightly decrease the performance of AxoSyslog, reducing the number of transferred messages during a given period.

Available in version 3.19 and later.

ca-dir()

Accepted values:	Directory name
Default:	none

Description: The 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 server authentication on the AxoSyslog clients. The AxoSyslog application uses the CA certificates in this directory to validate the certificate of the peer.

This option can be used together with the optional `ca-file()` option.

ca-file()

Accepted values:	File name
Default:	empty

Description: Optional. The name of a file that contains a set of trusted CA certificates in PEM format. The AxoSyslog application uses the CA certificates in this file to validate the certificate of the peer.

Example format in configuration:

```
ca-file("/etc/pki/tls/certs/ca-bundle.crt")
```

> **Note**
>
> **The `ca-file()` option can be used together with the `ca-dir()` option, and it is relevant when `peer-verify()` is set to other than `no` or `optional-untrusted`.**

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 AxoSyslog application uses this certificate to authenticate the AxoSyslog 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 AxoSyslog 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 AxoSyslog. 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 AxoSyslog receives TLS-encrypted connections, the order of ciphers set on the AxoSyslog server takes precedence over the client settings.

client-sigalgs()

Accepted values:	string [colon-separated list]
Default:	none

Description: A colon-separated list that specifies the supported signature algorithms associated with client authentication for TLSv1.2 and higher, for example, `RSA-PSS+SHA256:ed25519`.

- For servers, the value is used in the `signature_algorithms` field of a `CertificateRequest` message.
- For clients, it is used to determine which signature algorithm to use with the client certificate.

If `client-sigalgs()` is not set but `sigalgs()` is, then the values of `sigalgs()` are used.

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 AxoSyslog supports only DH parameter files in the PEM format. If you do not set this parameter, AxoSyslog 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")
```

extended-key-usage-verify()

Accepted values:	`yes` , `no`
Default:	`no`

Available in AxoSyslog 4.23 and later.

Description: If enabled, AxoSyslog checks that the certificate of the peer contains the appropriate value in its Extended Key Usage (EKU) field: clients should have Client Auth EKU values (`X509_PURPOSE_SSL_CLIENT`), while server should have Web Server Auth (`X509_PURPOSE_SSL_SERVER`).

If `extended-key-usage-verify()` is set to `yes` and the certificate of the peer doesn't have the appropriate Extended Key Usage value, AxoSyslog terminates the connection.

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 AxoSyslog application uses this private key and the matching certificate (set in the `cert-file()` option) to authenticate the AxoSyslog client on the destination server.

keylog-file()

Accepted values:	Filename
Default:	N/A

Description: This option enables saving TLS secrets (decryption keys) for a given source or destination, which can be used to decrypt data with, for example, Wireshark. The given path and name of a file will be used to save these secrets.

This option is only available with the following drivers:

- network
- syslog
- tcp
- tcp6

> **Warning**
>
> **Using `keylog-file()` makes TLS connections less secure by writing secret key materials into the given file. This option should only be enabled for debugging purposes and should be disabled after that. It is also recommended to delete the file after the debugging session is over.**

ocsp-stapling-verify

Accepted values:	`yes` , `no`
Default:	`no`

Available in AxoSyslog 4.0 and later.

Description: When OCSP stapling verification is enabled, AxoSyslog requests the server to send back its OCSP status. AxoSyslog verifies this status response using the trust store you have configured using the `ca-file()` , `ca-dir()` , or the `pkcs12-file()` options.

> **Note**
>
> **RFC 6961 multi-stapling and TLS 1.3-provided multiple responses are currently not validated, only the peer certificate is verified.**

Example configuration:

```
destination {

    network("example.com" transport(tls)
        tls(
            pkcs12-file("/path/to/test.p12")
            peer-verify(yes)
            ocsp-stapling-verify(yes)
        )
    );
};
```

openssl-conf-cmds()

Available in AxoSyslog version 4.0 and later.

IMPORTANT: `openssl-conf-cmds()` always has the highest priority, so it overrides any other options that can be found in the `tls()` section.

OpenSSL offers an alternative, software-independent configuration mechanism through the SSL_CONF_cmd interface for setting the various SSL_CTX and SSL options.

The order of operations within `openssl-conf-cmds()` is significant and the commands are executed in top-down order. This means that if the same option occurs multiple times, then the 'last one wins'. This is also true for options that can be set multiple ways (for example, cipher suites or protocols).

Example configuration:

```
tls(
    ca-dir("/etc/ca.d")
    key-file("/etc/cert.d/serverkey.pem")
    cert-file("/etc/cert.d/servercert.pem")
    peer-verify(yes)

    openssl-conf-cmds(
        # For system wide available cipher suites use: /usr/bin/openssl ciphers -v
        # For formatting rules see:
https://www.openssl.org/docs/man1.1.1/man3/SSL_CONF_cmd.html
        "CipherString" => "ECDHE-RSA-AES128-SHA",
# TLSv1.2 and bellow
        "CipherSuites" => "TLS_CHACHA20_POLY1305_SHA256:TLS_AES_256_GCM_SHA384",
# TLSv1.3+ (OpenSSl 1.1.1+)

        "Options" => "PrioritizeChaCha",
        "Protocol" => "-ALL,TLSv1.3",
    )
)
```

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).
- 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

Local peer-verify() setting	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 – AxoSyslog accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatches.

Warning

When validating a certificate, the entire certificate chain must be valid, including the CA certificate. If any certificate of the chain is invalid, AxoSyslog will reject the connection.

Starting with AxoSyslog 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 AxoSyslog configuration.

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

sigalgs()

Accepted values:	string [colon-separated list]
Default:	none

Description: A colon-separated list that specifies the supported signature algorithms (in order of decreasing preference) for TLSv1.2 and higher, for example, `RSA-PSS+SHA256:ed25519`. If this option is not set then all supported signature algorithms supported are permissible.

- For servers, it is used to determine which signature algorithms to support.
- For clients, this value is used directly for the supported signature algorithms extension.

sni()

Accepted values:	`yes` or `no`
Default:	`no`

Description: When set to `yes` in a destination that uses TLS encryption, this option enables Server Name Indication (also called Server Name Identification, SNI). The AxoSyslog sends the hostname or the IP address set in the destination to the server during the TLS handshake.

Available in AxoSyslog 3.24 and newer.

Example: Using Server Name Indication

The following destination sends the hostname of its destination during the TLS handshake.

```
destination demo_tls_destination_with_sni {
     network(
          "logserver.example.com" 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")
             sni(yes)
         )
     );
 };
```

ssl-options()

Accepted values:	comma-separated list of the following options: `no-sslv2` , `no-sslv3` , `no-tlsv1` , `no-tlsv11` , `no-tlsv12` , `no-tlsv13` , `none` , `ignore-hostname-mismatch` , `ignore-validity-period`
Default:	no-sslv2

Available in AxoSyslog 3.7 and newer.

Description: Sets the specified options of the SSL/TLS protocols. You can use it to disable specific protocol versions, and set other options. 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 AxoSyslog 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.

By specifying `ignore-hostname-mismatch` , you can ignore the subject name of a certificate during the validation process. This means that AxoSyslog checks only that the certificate itself is trusted by the current set of trust anchors (e.g. trusted CAs), and

ignores the mismatch between the targeted hostname and the certificate subject. `ignore-hostname-mismatch` is available in AxoSyslog 4.4 and newer.

By specifying `ignore-validity-period`, you can ignore the you can ignore the validity periods of certificates during the certificate validation process. `ignore-validity-period` is available in AxoSyslog 4.5 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)
         )
     );
 };
```

ssl-version()

Type:	string
Default:	None, uses the libcurl default

Available in AxoSyslog version 4.5.0 and later.

Description: Specifies the permitted SSL/TLS version. Possible values: `sslv2`, `sslv3`, `tlsv1`, `tlsv1_0`, `tlsv1_1`, `tlsv1_2`, `tlsv1_3`.

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-fingerprints()

Accepted values:	list of accepted fingerprints
Default:	none

Available in AxoSyslog version 4.26.0 and later.

Description: To accept connections only from hosts using certain certificates having specific fingerprints, list the fingerprints of the accepted certificates in this parameter. for example:

```
trusted-
fingerprints("SHA1:0C:EF:34:4D:0B:74:AE:03:72:9A:4E:68:AF:90:59:A9:EF:35:1F:AA",

"SHA512:15:B3:C5:96:48:5B:F6:20:C3:86:47:78:99:E1:2B:F2:C4:A6:93:AE:E8:0A:B3:F7:78:
39:66:B4:EF:4F:A5:47:2A:E0:4A:93:06:46:72:C0:15:6A:FC:59:10:25:37:60:E3:84:E9:EC:90
:30:12:F5:27:EA:22:1F:55:9B:3B:97")
```

> **CAUTION:**
>
> If the fingerprint of the peer is listed in `trusted-fingerprints()`, AxoSyslog accepts the certificate even if it's not valid.

To create the `trusted-fingerprints` string of a certificate:

1. Find the fingerprint of a certificate, for example, using the following command:

   ```
   openssl x509 -sha512 -in <certificate file in pem> -fingerprint -noout
   ```

   Sample output:

```
sha512
Fingerprint=27:42:63:34:F5:56:63:2B:AD:8F:BD:AD:29:17:94:7E:CB:17:7F:40:A2:FB:3
6:C1:4C:BA:6C:A7:E1:1C:A5:EE:79:E5:01:17:F2:98:F7:25:C9:FC:01:50:64:AB:5B:6C:D4
:44:60:1F:5A:9F:BC:0A:E6:0A:15:16:74:21:9E:A9
```

2. Replace the `Fingerprint=` part with a colon (`:`), for example:
   `sha512:27:42:63:34:F5:56:63:2B:AD:8F:BD:AD:29:17:94:7E:CB:17:7F:40:A2:FB:36:C1:4C:BA:6C:A7:E1:1C:A5:EE:79:E5:01:17:F2:98:F7:25:C9:FC:01:50:64:AB:5B:6C:D4:44:60:1F:5A:9F:BC:0A:E6:0A:15:16:74:21:9E:A9`

   The fingerprinting method is the word before the first colon. The name of the fingerprinting methods must match an OpenSSL-supported digest algorithm, which are (for OpenSSL 3.0.10): `blake2b512, blake2s256, md4, md5, rmd160, sha1, sha224, sha256, sha3-224, sha3-256, sha3-384, sha3-512, sha384, sha512, sha512-224, sha512-256, shake128, shake256, sm3`

> **Note**
>
> When using the `trusted-fingerprints()` and `trusted-dn()` parameters, note the following:
>
> - First, the `trusted-fingerprints()` parameter is checked. If the fingerprint of the peer is listed, the certificate validation is skipped.
> - If the fingerprint of the peer is not listed in the `trusted-fingerprints()` 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.

AxoSyslog automatically adds the fingerprint of the peer to the `${.tls.x509_fp}` name-value pair.

trusted-keys() (DEPRECATED)

Accepted values:	list of accepted SHA-1 fingerprints
Default:	none

Deprecated in AxoSyslog version 4.26.0. Use `trusted-fingerprints()` instead.

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.

Starting with version 4.8.1, if `trusted-keys()` is set, AxoSyslog automatically adds the key fingerprint of the peer to the `${.tls.x509_fp}` name-value pair.

13 template and rewrite: Format, modify, and manipulate log messages

Reformat messages with templates and macros, change message content with rewrite rules, and look up the supported regular expression types.

> **FilterX or classic filters, parsers, and rewrite rules?**
>
> **FilterX replaces filters, parsers, and rewrite rules with a single typed language that can also handle nested JSON and other complex data. You don't have to migrate: the classic blocks keep working, and you can use both in the same log path. To convert your existing configuration, see Update to FilterX.**

This chapter explains the methods that you can use to customize, reformat, and modify log messages using AxoSyslog.

- 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.
- Modify 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 AxoSyslog objects like filters and rewrite rules.

13.1 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.
- Modify 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 AxoSyslog lists the different types of macros available in AxoSyslog.
- Using template functions explains what template functions are and how to use them.
- Template functions of AxoSyslog lists the template functions available in AxoSyslog.

13.1.1 Formatting messages, filenames, directories, and tablenames

The AxoSyslog 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 the message was received. 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 AxoSyslog, see Macros of AxoSyslog.
- 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.

13.1.2 Templates and macros

The AxoSyslog 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 AxoSyslog, see Macros of AxoSyslog. 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)`). Enabling escaping causes AxoSyslog to escape the `'`, `"`, and backslash characters from the messages. This behavior is useful when the messages are passed to an application that cannot handle escaped characters properly.

> Note: Starting with AxoSyslog version 4.5, `template-escape(yes)` escapes the top-level template function in case of nested template functions.

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.

Macro names are case-sensitive, that is, `"$message"` and `"$MESSAGE"` are not the same.

To use a literal `$` character in a template, you have to escape it. In AxoSyslog 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, AxoSyslog 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.

13.1.3 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.

  > **Warning**
  >
  > **To use the `S_` macros, the `keep-timestamp()` option must be enabled (this is the default behavior of AxoSyslog).**

- `R_` prefix, for example, `${R_DATE}`: `${R_DATE}` is the date when AxoSyslog has received the message.
- `C_` prefix, for example, `${C_DATE}`: `${C_DATE}` is the current date, that is when AxoSyslog 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. Alternatively, you can modify the timezone of the message using timezone-specific rewrite rules. For details, see Rewrite the timezone of a message.

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`
- `ISOWEEK`

- `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`

13.1.4 Hard versus soft macros

Hard macros contain data that is directly derived from the log message, for example, the `${MONTH}` macro derives its value from the timestamp. Hard macros are read-only. Soft macros (sometimes also called name-value pairs) are either built-in macros automatically generated from the log message (for example, `${HOST}`), or custom user-created macros generated by using the pattern database or a CSV-parser. In contrast to hard macros, soft macros are writable and can be modified within AxoSyslog, 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 AxoSyslog are hard macros and cannot be modified: `BSDTAG`, `CONTEXT_ID`, `DATE`, `DAY`, `FACILITY_NUM`, `FACILITY`, `FULLDATE`, `HOUR`, `IP_PROTO`, `ISODATE`, `ISOWEEK`, `LEVEL_NUM`, `LEVEL`, `MIN`, `MONTH_ABBREV`, `MONTH_NAME`, `MONTH`, `MONTH_WEEK`, `PRIORITY`, `PRI`, `RCPTID`, `SDATA`, `SEC`, `SEQNUM`, `SOURCEIP`, `STAMP`, `TAG`, `TAGS`, `TRANSPORT`, `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).

Note that you can modify the timezone of the message, and change the timezone-related macros that way. For details, see Rewrite the timezone of a message.

13.1.5 Macros of AxoSyslog

The following macros are available in AxoSyslog templates.

> **Warning**
>
> These macros are available when AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

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 AxoSyslog 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`.

See Date-related macros.

DAY, C_DAY, R_DAY, S_DAY

Description: The day the message was sent.

See Date-related macros.

DESTIP

Description: When used, the output specifies the local IP address of the source from which the message originates.

For an example use case when using the macro is recommended, see Example use case: using the $DESTIP, the $DESTPORT, and the $PROTO macros.

DESTPORT

Description: When used, the output specifies the local port of the source from which the message originates.

For an example use case when using the macro is recommended, see Example use case: using the $DESTIP, the $DESTPORT, and the $PROTO macros.

FACILITY

Description: The name of the facility (for example, `kern`) that sent the message. For the list of facility codes, see The PRI message part .

FACILITY_NUM

Description: The numerical code of the facility (for example, `0`) that sent the message. For the list of facility codes, see The PRI message part .

FILE_NAME

Description: Name of the log file (including its path) from where AxoSyslog received the message (only available if AxoSyslog 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`.

See Date-related macros.

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 `${FULLHOST}` macro will be the DNS hostname of the host that sent the message to AxoSyslog (that is, the DNS hostname of the last hop). In this case the `${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 AxoSyslog, see Using name resolution in syslog-ng.

FULLHOST_FROM

Description: The FQDN of the host that sent the message to AxoSyslog as resolved by AxoSyslog using DNS. If the message traverses several hosts, this is the last host in the chain.

The AxoSyslog application uses the following procedure to determine the value of the `$FULLHOST_FROM` macro:

1. The AxoSyslog application takes the IP address of the host sending the message.
2. If the `use-dns()` option is enabled, AxoSyslog 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 AxoSyslog, see Using name resolution in syslog-ng.

HOUR, C_HOUR, R_HOUR, S_HOUR

Description: The hour of day the message was sent.

See Date-related macros.

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 AxoSyslog 3.4 and later.

See Date-related macros.

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 AxoSyslog (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 AxoSyslog, see Using name resolution in syslog-ng.

HOST_FROM

Description: The FQDN of the host that sent the message to AxoSyslog as resolved by AxoSyslog using DNS. If the message traverses several hosts, this is the last host in the chain.

The AxoSyslog application uses the following procedure to determine the value of the `$HOST_FROM` macro:

1. The AxoSyslog application takes the IP address of the host sending the message.
2. If the `use-dns()` option is enabled, AxoSyslog 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 AxoSyslog, see Using name resolution in syslog-ng.

IP_PROTO

Available in AxoSyslog version 4.5 and later.

The IP protocol version used to retrieve or receive the message. Contains either “4” to indicate IPv4 and “6” to indicate IPv6. IPv4 addresses mapped to IPv6 count as IPv4. If the message has no source IP address (for example, it comes from a local source), the value is “0”.

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 AxoSyslog can produce fractions of a second (for example, milliseconds) in the timestamp by using the `frac-digits()` global or per-destination option.

See Date-related macros.

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

ISOWEEK, C_ISOWEEK, R_ISOWEEK, S_ISOWEEK

Description: The number of week according to the ISO 8601 standard. Note that the `${WEEK}` macro that has been available in returns a non-standard week number that can differ from the value returned by the `${ISOWEEK}` macro.

See Date-related macros.

Available in 3.24 and later.

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 AxoSyslog.

- In version 3.24 and later: the `${LOGHOST}` macro returns the fully-qualified domain name (FQDN) only if the `use-fqdn()` option is set to yes, and the hostname otherwise.

- In earlier versions: the `${LOGHOST}` macro returns the fully-qualified domain name (FQDN).

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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

The `${MSG}` macro is an alias of the `${MESSAGE}` macro: using `${MSG}` in AxoSyslog is equivalent to ${MESSAGE}.

Note that before AxoSyslog version 3.0, the `${MESSAGE}` macro included the program name and the pid. In AxoSyslog 3.0, the `${MESSAGE}` macro became equivalent with the `${MSGONLY}` macro.

MIN, C_MIN, R_MIN, S_MIN

Description: The minute the message was sent.

See Date-related macros.

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.

See Date-related macros.

MONTH_ABBREV, C_MONTH_ABBREV, R_MONTH_ABBREV, S_MONTH_ABBREV

Description: The English abbreviation of the month name (3 letters).

See Date-related macros.

MONTH_NAME, C_MONTH_NAME, R_MONTH_NAME, S_MONTH_NAME

Description: The English name of the month name.

See Date-related macros.

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.

See Date-related macros.

MSEC, C_MSEC, R_MSEC, S_MSEC

Description: The millisecond the message was sent.

See Date-related macros.

Available in AxoSyslog version 3.4 and later.

MQTT_TOPIC

Description: The `mqtt()` source automatically sets the `${MQTT_TOPIC}` name-value pair for the messages it receives. This is useful when the name of the topic contains MQTT wildcards (`$` , `+` , `#`).

Available in AxoSyslog version 4.7 and later.

MSG

The `${MSG}` macro is an alias of the `${MESSAGE}` macro, using `${MSG}` in AxoSyslog is equivalent to `${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.

MSGFORMAT

Available in AxoSyslog version 4.8.1 and later.

Description: Stores the original format of the incoming message. Possible values:

- `linux:devkmsg` : Linux kernel message.
- `linux:pacct` : Linux process accounting log format.
- `raw` : AxoSyslog didn't parse the message, for example, because the `no-parse` flag was set.
- `syslog:rfc3164` : Syslog message formatted as RFC3164.
- `syslog:rfc5424` : Syslog message formatted as RFC5424.

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, AxoSyslog 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 AxoSyslog 3.0, the following macros are equivalent: `${MSGONLY}` , `${MSG}` , `${MESSAGE}` . For consistency, use the `${MESSAGE}` macro. For details, see MESSAGE .

PEERIP

Available in AxoSyslog 4.11 and later. This macro is available when using the `network()` or the `syslog()` source, or when using the Webhook with the `proxy-header()` option set.

Description: IP address of the host that sent the message to AxoSyslog. In most cases, the `${PEERIP}` and `${PEERPORT}` values are identical to `${SOURCEIP}` and `${SOURCEPORT}` . However, when dealing with proxied protocols, `${PEERIP}` and `${PEERPORT}` contain the proxy's address and port, while `${SOURCEIP}` and `${SOURCEPORT}` contain the original source of the message.

PEERPORT

Available in AxoSyslog 4.11 and later. This macro is available when using the `network()` or the `syslog()` source.

Description: The port of the host that sent the message to AxoSyslog. For details, see `${PEERIP}` .

PID

Description: The PID of the program sending the message.

PRI

Description: The combined facility and severity value (`facility * 8 + severity`), encoded as a 2 or 3 digit decimal number as it is present in the `<N>` header of syslog messages. This encodes both facility and severity, unlike `${LEVEL_NUM}` , which is the severity alone. For the facility and severity code tables and the priority formula, see The PRI message part .

PRIORITY or LEVEL

Description: The priority (also called severity or level) of the message as a text label, for example, `err` , `warning` , `notice` , or `info` . The `${PRIORITY}` , `${LEVEL}` , and `${SEVERITY}` macros are aliases and return the same value. For the numeric representation, use the `${LEVEL_NUM}` macro. For what each severity means, see syslog Message Severities .

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.

PROTO

Description: Returns the Assigned Internet Protocol Number of the protocol used on the source from which the message originates: `6` for TCP-based sources, and `17` for UDP-based sources. See also the `${PROTO_NAME` macro .

For an example use case when using the macro is recommended, see Example use case: using the $DESTIP, the $DESTPORT, and the $PROTO macros.

PROTO_NAME

Available in AxoSyslog version 4.16 and newer.

Description: Returns the name of the protocol (`tcp` or `udp`) AxoSyslog received the message on, if it was a TCP or UDP based protocol. This corresponds to the values `6` and `17` of the `${PROTO}` macro.

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 AxoSyslog received the message using the `default-network-drivers()` source , or the source receiving the message has the `store-raw-message` flag set.

RAWMSG_SIZE

Available in AxoSyslog version 4.2 and newer.

Description: The original size of the incoming message in bytes. Might not be available for every source driver.

RCPTID

Description: When the `use-rcptid` global option is set to `yes` , AxoSyslog 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 64-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 AxoSyslog is restarted, but not when reloaded.

SDATA, .SDATA.SDID.SDNAME

Description: The AxoSyslog 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.

See Date-related macros.

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 AxoSyslog receives a message via the IETF-syslog protocol that includes a sequence ID, this ID is automatically available in the `${SEQNUM}` macro.
- If the message is a Cisco IOS log message using the extended timestamp format, then AxoSyslog stores the sequence number from the message in this macro. If you forward this message the IETF-syslog protocol, AxoSyslog includes the sequence number received from the Cisco device in the `${.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), AxoSyslog 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 AxoSyslog is reloaded, but it is reset when AxoSyslog 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 AxoSyslog receives, use the RCPTID macro.**

SOURCE

Description: The identifier of the source statement in the AxoSyslog configuration file that received the message. For example, if AxoSyslog 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 AxoSyslog. (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.

SOURCEPORT

Available in AxoSyslog 4.10 and later.

Description: The source port of the host that sent the message to AxoSyslog.

STAMP, R_STAMP, S_STAMP

Description: A timestamp formatted according to the `ts-format()` global or per-destination option.

See Date-related macros.

SYSUPTIME

Description: The time elapsed since the AxoSyslog instance was started (that is, the uptime of the AxoSyslog process). The value of this macro is an integer containing the time in 1/100th of the second.

Available in AxoSyslog 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.

TRANSPORT

Available in AxoSyslog version 4.5 and later.

AxoSyslog automatically populates this name-value pair with the “transport” mechanism used to retrieve or receive the message. The exact value depends on the source driver that received the message. Currently the following values are implemented:

- BSD syslog drivers `tcp()` , `udp()` & `network()`
  - `rfc3164+tls`
  - `rfc3164+tcp`
  - `rfc3164+udp`
  - `rfc3164+proxied-tls`
  - `rfc3164+<custom logproto like altp>`
- UNIX domain drivers `unix-dgram()` , `unix-stream()`
  - `unix-stream`
  - `unix-dgram`
- RFC5424-style syslog `syslog()` :
  - `rfc5426` : syslog over udp
  - `rfc5425` : syslog over tls
  - `rfc6587` : syslog over tcp
  - `rfc5424+<custom logproto like altp>` : syslog over a logproto plugin
- Other drivers:
  - `otlp` : `otel()` driver
  - `mqtt` : `mqtt()` driver
  - `hypr-api` : `hypr-audit-source()` driver

- Locally created logs (in version 4.7 and newer):
  - `local+unix-stream`
  - `local+unix-dgram`
  - `local+file`
  - `local+pipe`
  - `local+program`
  - `local+devkmsg`
  - `local+journal`
  - `local+afstreams`
  - `local+openbsd` (logs from the `openbsd()` driver)

TZ, C_TZ, R_TZ, S_TZ

Description: An alias of the ${TZOFFSET} macro.

See Date-related macros.

TZOFFSET, C_TZOFFSET, R_TZOFFSET, S_TZOFFSET

Description: The time-zone as hour offset from GMT, for example: `-07:00` . In version 1.6.x this used to be `-0700` but as `${ISODATE}` requires the colon it was added to `${TZOFFSET}` as well.

See Date-related macros.

UNIXTIME, C_UNIXTIME, R_UNIXTIME, S_UNIXTIME

Description: Standard UNIX timestamp, represented as the number of seconds since `1970-01-01T00:00:00` .

See Date-related macros.

.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 AxoSyslog 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.
- `.tls.x509_fp` : The key fingerprint of the peer, if the `trusted-keys()` option or `trusted-fingerprints()` option is used. Available in version 4.8.1 and later.

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 AxoSyslog version 3.7 and later.

USEC, C_USEC, R_USEC, S_USEC

Description: The microsecond the message was sent.

See Date-related macros.

Available in AxoSyslog version 3.4 and later.

YEAR, C_YEAR, R_YEAR, S_YEAR

Description: The year the message was sent.

See Date-related macros.

WEEK, C_WEEK, R_WEEK, S_WEEK

Description: The week number of the year, prefixed with a zero for the first nine weeks of the year. (The first Monday in the year marks the first week.)

See Date-related macros.

See also ISOWEEK, C_ISOWEEK, R_ISOWEEK, S_ISOWEEK .

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` .

See Date-related macros.

WEEK_DAY, C_WEEK_DAY, R_WEEK_DAY, S_WEEK_DAY

Description: The day of the week as a numerical value (1-7).

See Date-related macros.

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` .

See Date-related macros.

WEEK_DAY_NAME, C_WEEK_DAY_NAME, R_WEEK_DAY_NAME, S_WEEK_DAY_NAME

Description: The English name of the day.

See Date-related macros.

13.1.5.1 Example use case: using the $DESTIP, the $DESTPORT, and the $PROTO macros

This section describes scenarios when Axoflow recommends using the $DESTIP, the $DESTPORT, and the $PROTO macros.

Using the $DESTIP, the $DESTPORT, and the $PROTO macros is relevant when multiple sources are configured to receive messages on the AxoSyslog side. In this case, the hostname and IP address on the sender's side and the AxoSyslog side is the same, and at a later point in the pipeline, AxoSyslog can not by default specify which source received the message. The $DESTIP, the $DESTPORT, and the $PROTO macros solve this issue by specifying the local IP address and local port of the original message source, and the protocol used on the original message source on the AxoSyslog side.

When to use the $DESTIP, the $DESTPORT, and the $PROTO or $PROTO_NAME macros

Axoflow recommends using the $DESTIP, the $DESTPORT, and the $PROTO macros in either of the following scenarios:

- Your appliance sends out log messages through both UDP and TCP.
- The format of the UDP log messages and the TCP log messages is different, and instead of using complex filters, you want to capture either of them, preferably with the simplest possible filter.
- The IP addresses on the sender's side and the AxoSyslog side are the same, therefore the `netmask()` option doesn't work in your configuration.
- The hostnames on the sender's side and the AxoSyslog side are the same, therefore the `host()` option doesn't work in your configuration.

Macros: $DESTIP, $DESTPORT, and $PROTO

To solve either of the challenges listed previously, AxoSyslog supports the following macros that you can include in your configuration:

- `$DESTIP`
- `$DESTPORT`
- `$PROTO`
- `$PROTO_NAME`

Configuration and output

The following configuration example illustrates how you can use the $DESTIP, the $DESTPORT, and the $PROTO macros in your AxoSyslog configuration.

The $DESTIP, the $DESTPORT, and the $PROTO macros in your AxoSyslog configuration:

```
log{
  source{
    network(localip(10.12.15.215) port(5555) transport(udp));
  };

destination {
  file("/dev/stdout" template("destip=$DESTIP destport=$DESTPORT proto=$PROTO\n"));
  };
};
```

With these configuration settings, the macros will specify the local IP, the local port, and the protocol information of the source from which the message originates as follows:

```
destip=10.12.15.215 destport=5555 proto=17
```

Starting with version 4.16, you can also use the `${PROTO_NAME}` macro, which returns `tcp` or `udp`, depending on the protocol used to receive the message.

13.1.6 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 AxoSyslog. Template functions use the following syntax:

```
$(function-name parameter1 parameter2 parameter3 ...)
```

For example, the `$(echo)` template function simply returns the value of the macro it receives as a parameter, thus `$(echo ${HOST})` is equivalent to `${HOST}`.

The parameters of template functions are separated by a whitespace character. A template function can have maximum 64 parameters. If you want to use a longer string or multiple macros as a single parameter, enclose the parameter in double-quotes or apostrophes. For example:

```
$(echo "${HOST} ${PROGRAM} ${PID}")
```

Template functions can be nested into each other, so the parameter of a template function can be another template function, like:

```
$(echo $(echo ${HOST}))
```

For details on the available template functions, see the descriptions of the individual template functions in Template functions of AxoSyslog.

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}";
```

13.1.7 Template functions of AxoSyslog

The following template functions are available in AxoSyslog.

base64-encode

Syntax:

```
$(base64-encode argument)
```

Description: You can use the `base64-encode` template function to base64-encode strings and macros. The template function can receive multiple parameters (maximum 64). In this case, AxoSyslog joins the parameters into a single string and encodes this string. For example, `$(base64-encode string1 string2)` is equivalent to `$(base64-encode string1string2)` .

Available in AxoSyslog version 3.18 and later.

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 AxoSyslog version 3.10 and later.

ceil

Syntax:

```
$(ceil argument)
```

Description: Rounds a floating-point number upwards to the nearest integer. For example, `$(ceil 1.5)` is 2, `$(ceil -1.5)` is -1. See also the `floor` and `round` template functions.

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 email recipients from

postfix messages.

Available in AxoSyslog 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 AxoSyslog 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 AxoSyslog 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 AxoSyslog 3.5 and later.

explode

Syntax:

```
$(explode <separator> <string1> <string2> ...)
```

Description: Turns a string separated by a specific character into a list. You can also use the implode template function, which turns a list into a string combining the pieces together with a separator. Available in AxoSyslog 3.21 and later.

Example: Using the explode template function

The following configuration example turns strings into a list. If there are several strings, AxoSyslog looks for a separator within each individual string. For example, `string 2` is separated as `string, 2` in the example below:

Configuration	Result
`$(explode ';' string1;string 2;string3;string4)`	`"string1,string,2,string3,string4"`

Enclose the strings in double-quotes or apostrophes and `string 2` is separated as shown below:

Configuration	Result

`$(explode ' ' 'string1 string 2 string3 string4 string5')`	`"string1,string 2,string3,string4,string5"`

The following examples replace the separator `';'` character with a `','` character:

Configuration	Result
`$(implode ',' $(explode ';' 'string1;string2;string3'))`	`"string1,string2,string3"`
`$(explode ';' 'string1;string2;string3;string4;string5')`	`"string1,string2,string3,string4,string5"`

filter

Syntax: `$(filter <filter-expression> <list>)`

Description: Runs the filter expression on each element of a given list, and returns only those list elements that meet the requirements of the filter expression. The current value is referred by `$_` , similarly to the `map` template function .

> **Note**
>
> **You can use macros, logical expressions, and template functions inside the expression.**

Available in AxoSyslog version 3.30 and later.

Example: using the filter template function in your configuration

When used in configuration as seen in the example, the `filter` template function filters even numbers from an input list of `0` , `1` , `2` and `3` :

```
log {
    source { example-msg-generator(num(1) values(INPUT => "0,1,2,3")); };
    destination {
        file("/dev/stdout"
            template("$(filter ('$(% $_ 2)' eq '0') $INPUT)\n)")
        );
    };
};
```

The returned values are `0` and `2`.

Parameters

- `<filter-expression>`: Mandatory parameter. The `<filter-expression>` parameter can be:
    - a comparison
    - a filter
    - a logical expression built from filters (using `and`, `or`, and `not`)

    > **Note**
    > - When using the `<filter-expression>` parameter, you can refer other template functions, or use macros.
    > - To refer to the variable bound to the current element of the list, use `$_ macro`.

    The following examples illustrate several ways that you can use a single filter, or a logical expression built from several filters.

    ```
    ('1' == '1')
    ('$_' le '1')
    ('$(% $_ 2)' eq '0')
    ('$_' le '1') and ('$(% $_ 2)' eq '0')
    ```

- `list`: Mandatory parameter. A AxoSyslog list.

format-cef-extension

AxoSyslog 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 AxoSyslog 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:

- Set the `on-error` global option to `drop-property` , otherwise if the name of a name-value pair includes an invalid character, AxoSyslog 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 AxoSyslog GitHub repository .

> **Note**
>
> **To use the `format-cim()` template function, AxoSyslog must be compiled with JSON support. To see if your AxoSyslog binary was compiled with JSON support, execute the `syslog-ng --version` command.**

format-date

Syntax: `$(format-date [options] format-string [timestamp])`

Available in AxoSyslog 4.1 and later.

Description: The `$(format-date)` template function takes a timestamp in the DATETIME representation and formats it according to an strftime() format string . The DATETIME representation is a UNIX timestamp formatted as a decimal number, with an optional fractional part, where the seconds and the fraction of seconds are separated by a dot.

If the timestamp argument is missing, the timestamp of the message is used.

Options: `--time-zone <TZstring>` : Override timezone of the original timestamp

For example: `$(format-date --time-zone PST8PDT %Y-%m-%dT%H:%M:%S 1667500613)` corresponds to the following format and sets the timezone to Pacific Standard Time, Daylight Saving: “2022-11-03T11:36:53”

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 following is a sample log message in EWMM format.

```
<34>1 2026-10-11T22:14:15+00:00 my-host @syslog-ng - - -
{"PROGRAM":"su","MESSAGE":"'su root' failed for lonvick on
/dev/pts/8","HOST":"mymachine","HOST_FROM":"my-host","._TAGS":
[".source.s_network"]}
```

> **Note**
>
> To use this template function, the `scl.conf` file must be included in your AxoSyslog configuration:
>
> ```
> @include "scl.conf"
> ```

format-flat-json

Syntax: `$(format-flat-json parameters)`

Description: The `format-flat-json` template function is identical to the `format-json` template function, but nested JSON objects are flattened in the output. If you have to forward your log messages in JSON format, but the receiving application cannot handle nested JSON objects, use the `format-flat-json` template function.

Example: Flattened JSON output

The following example shows the difference between nested and flattened JSON objects.

- The output of `$(format-json a.b.c=1)` is a nested JSON object (whitespace added for better readability):

```
{
    "a": {
        "b": {
        "c": "1"
        }
    }
}
```

- The output of `$(format-flat-json a.b.c=1)` is a flattened JSON object (whitespace added for better readability):

```
{
    "a.b.c": "1"
}
```

For details on formatting log messages into JSON format, see `format-json` .

format-gelf

Syntax: `$(format-gelf)`

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

Description: Available in AxoSyslog 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 AxoSyslog 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.

If the value-pair includes type information `format-json` can propagate it to the next component of the log path. 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**
>
> Prior to version 4.0, AxoSyslog handled every data as strings, and allowed you to convert the strings into other types of data that only certain destinations data formats supported.
>
> Starting with AxoSyslog 4.0, each name-value pair is a (name, type, value) triplet, and several components of AxoSyslog have typing support. For details, see Components that support data types.

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"}}}`).
>
> To retain the starting dot, use the `--leave-initial-dot` flag, for example:
>
> ```
> $(format-json --leave-initial-dot .SDATA.meta.sequenceID)
> ```

If you have to forward your log messages in JSON format, but the receiving application cannot handle nested JSON objects, use the `format-flat-json` template function. For details, see `format-flat-json` .

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 AxoSyslog 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})
```

This feature requires a separate module. Install the `axosyslog-mod-geoip2` package on Debian/Ubuntu , or the `axosyslog-geoip` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Description: This template function extracts specific fields from the mmdb database using the `--field` parameter. If you omit this parameter, it returns the 2-letter country code of any IPv4/IPv6 address or host.

> **Note**
>
> **This template function is available only if AxoSyslog has been compiled with geoip2 support. To enable it, use the `--enable-geoip` compiling option.**

To retrieve additional GeoIP information, see Looking up GeoIP2 data from IP addresses.

Starting with version 3.24, AxoSyslog tries to automatically detect the location of the database. If that is successful, the `database()` option is not mandatory.

getent

Syntax: `$(getent)`

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-getent` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Description: Available in AxoSyslog 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)`

On Debian/Ubuntu, this feature is available in a separate module: install the `axosyslog-mod-graphite` package. On RHEL and compatible distributions, it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

Description: Available in AxoSyslog 3.6 and later. This template function converts value-pairs from the incoming message to the Graphite plain text protocol format.

For details on selecting value-pairs in AxoSyslog and for possibilities to specify which information to convert to Graphite plain text protocol format, see Structuring macros, metadata, and other value-pairs. Note that the syntax of `graphite-output` is different from the syntax of `value-pairs()` : `graphite-output` uses a the command-line syntax used in the format-json template function .

Example: Using the graphite-output template function

The following configuration example shows, how to send value-pairs with names starting with “ `vmstat.` ” to Graphite running on `localhost` , port `2003` :

```
   destination d_graphite {
        network( host("localhost") port(2003) template("$(graphite-output --key
vmstat.*)"));
    };
```

grep

Syntax: `$(grep condition value-to-select)`

Description: The `grep` template function can search a message context when correlating messages (for example, when you use a pattern database or the grouping-by parser). The `context-lookup` template function requires a condition (a filter or a string), and returns a specific macro or template of the matching message (for example, the ${MESSAGE} field of the message).

Example: Using the grep template function

The following example selects the message of the context that has a `username` name-value pair with the `root` value, and returns the value of the `auth_method` name-value pair.

```
   $(grep ("${username}" == "root") ${auth_method})
```

You can to specify multiple name-value pairs as parameters, separated with commas. If multiple messages match the condition of `grep` , these will be returned also separated by commas. This can be used for example, to collect the email recipients from postfix messages.

hash

Syntax: `$(<method> [opts] $arg1 $arg2 $arg3...)`

Options: `--length N, -l N`

Truncate the hash to the first N characters.

Description: Calculates a hash of the string or macro received as argument using the specified hashing method. If you specify multiple arguments, effectively you receive the hash of the first argument salted with the subsequent arguments.

`<method>` can be one of md5, md4, sha1, sha256, sha512 and "hash", which is equivalent to md5. Macros are expected as arguments, and they are concatenated without the use of additional characters.

The md4 `<method>` is deprecated.

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 AxoSyslog has been compiled with the `--enable-ssl` compile option and the `tfhash` module has been loaded.
>
> By default, AxoSyslog 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>
```

implode

Syntax: `$(implode <separator> <string1>, <string2>, ...)`

Description: Turns a list into a string combining the pieces together with a separator. You can also use the explode template function, which turns a string separated by a specific character into a list. Available in AxoSyslog 3.21 and later.

Example: Using the implode template function

The following configuration example shows how you can use the `implode` template to turn a list into a string:

Configuration	Result
`$(implode ' ' 'string1,string2,string3,string4,string5')`	`"string1 string2 string3 string4 string5"`

You can also use a $(list-*) template function to further manipulate the list. The following example returns the first three elements of the list:

Configuration	Result
`$(implode ' ' $(list-slice :3 string1,string2,string3,string4,string5))`	`"string1 string2 string3"`

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, AxoSyslog 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 AxoSyslog. 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 AxoSyslog 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-search

Syntax: `$(list-search [OPTIONS] <pattern> ${list})`

Description: The `list-search` template function searches the elements of `${list}` starting at the specified `start_index`, then returns the index of the first match of `<pattern> within` ${list}`.

> **Note**
>
> **Indexing is 0-based. If `<pattern>` is not found, the function returns an empty string.**

Options:

- `--mode MODE` : Matching mode, with the following possible values: `literal` (default), `prefix` , `substring` , `glob` , `pcre`
- `--start-index N` : Skips N elements in the `${list}`

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.

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` .

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 AxoSyslog 3.5 and later.

map

Syntax: `$(map template list)`

Description: Returns with a list that contains the results of applying a template function for each elements of a list.

Available in AxoSyslog version 3.28 and later.

Parameters:

- `template` : Mandatory. This template function is applied for each elements of the list. Use `$_ macro` to refer to the current list element.
- `list` : Mandatory. A list, or template.

Example:

When used in configuration as seen in the example, the `map` template function adds one to each element of a list:

```
log {
   source { example-msg-generator(num(1) values(LST => "0,1,2")); };
   destination {
     file("/dev/stdout"template('$(map "$(+ 1 $_)" $LST)')
     );
   };
 };
```

The returned values are `1`, `2`, and `3`.

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>}");
```

Starting with AxoSyslog version 3.22 and later, the numerical operators support floating-point values. They behave like the operators in the C programming language:

- If both operands are integers, they return an integer.
- If any of the operands is a floating-point number, they return a floating-point result.

For example:

```
   $(/ 3 2) # Both operands are integers, so the result is 1
    # One of the operands is a floating point number, so the result is also
floating-point
    $(/ 3.0 2) # = 1.500000
    $(/ 3 2.0) # = 1.500000
    $(/ 3.0 2.0) # = 1.500000
```

To round floating-point numbers, you can use the `ceil`, `floor`, and `round` template functions.

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: `XXXXXXXXXmymessage`

```
$(padding ${MESSAGE} 10 foo)
```

Output: `foofoofoofmymessage`

python

Syntax:

```
$(python <name-of-the-python-method-to-use> <arguments-of-the-method>)
```

This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Description: This template function enables you to write a custom template function in Python. You can define a Python block in your AxoSyslog configuration file, define one or more Python functions in it, and use the methods as template functions. If you use a Python block, AxoSyslog embeds a Python interpreter to process the messages.

The following points apply to using Python blocks in AxoSyslog in general:

- Python parsers and template functions are available in AxoSyslog version 3.10 and later.

  Python destinations and sources are available in AxoSyslog version 3.18 and later.

- Supported Python versions: 2.7 and 3.4+ (if you are using pre-built binaries, check the dependencies of the package to find out which Python version it was compiled with).

- The Python block must be a top-level block in the AxoSyslog configuration file.

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

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

- The Python object is initiated every time when AxoSyslog is started or reloaded.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

- The Python block can contain multiple Python functions.
- Using Python code in AxoSyslog can significantly decrease the performance of AxoSyslog, especially if the Python code is slow. In general, the features of AxoSyslog are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The AxoSyslog application does not do any of this.
- Python error messages are available in the `internal()` source of AxoSyslog.
- You can access the name-value pairs of AxoSyslog directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of AxoSyslog. For details, see Logging from your Python code.

The following points apply to Python parsers.

- The first argument in the definition of the Python function is the actual log message. This is implicitly passed to the function, you do not have to use it in the template function.
- The value of the template function is return value of the Python function.
- To reference a name-value pair or a macro in the Python 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: 4.29

 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: 4.29

 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 AxoSyslog 3.5 and later.

round

Syntax: `$(round argument)`

Description: Rounds a floating-point number to the nearest integer. For example, `$(round 1.5)` is 2. See also the `ceil` and `floor` template functions.

This template function has an optional second argument that sets the precision of rounding. The default is 0 (output a natural number), but values up to 20 are accepted. For example, `$(round 2.123456 4)` is 2.1235.

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>")`

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-stardate` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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);
    };
```

tag

Available in AxoSyslog version 4.6 and later.

Syntax:

```
$(tag <name-of-the-tag> <value-if-set> <value-if-unset>)
```

Description: Adds bit-like tags to the messages.

- If you do not set the `value-if-set` and `value-if-unset` arguments, the `$(tag)` template function acts as a boolean and expands to `0` or `1`, depending on whether the message has the specified tag set.
- If the `value-if-set` and `value-if-unset` arguments are set, `$(tag)` returns a string: the second argument (`<value-if-set>`) if the message has `<tag>`, and the third argument (`<value-if-unset>`) if the message doesn't have `<tag>`.

tags-head

Available in AxoSyslog version 4.7 and later.

Syntax:

```
$(tags-head <name-of-first-tag> <name-of-second-tag> ... )
```

Description: This template function accepts multiple tag names, and returns the first one that is set.

Example config:

```
# resolves to "bar" if "bar" tag is set, but "foo" is not
template("$(tags-head foo bar baz)")
```

template

Syntax:

```
$(template <template-name>)
$(template $<dynamic-template-name>)
$(template $<dynamic-template-name> '<optional-fallback-template>')
```

Description: This template function looks up the `<template-name>` in the configuration and uses that to format its result. The referenced template can be static or dynamic. For static templates, AxoSyslog resolves the template when it starts, or when the configuration is reloaded. For dynamic templates, the results are resolved runtime (for dynamic templates, the template name contains at least one '$' character). For example, the name of the template to be invoked can be extracted from the message, or from a name-value pair set using the `add-contextual-data()` feature.

For dynamic templates, you can set an optional second template. This second template will be the results of the template function if resolving the dynamic template fails for some reason. For example:

```
    $(template ${my-dynamic-template} '$DATE $HOST $MSGHDR$MSG\n')
```

Available in AxoSyslog 3.22 and later.

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 AxoSyslog 3.5 and later.

url-decode

Syntax: `$(url-decode <string-pr-macro-1> <string-pr-macro-2> ... )`

Description: You can use the `url-decode` template function to decode url-encoded strings and macros. For example, `$(url-decode %3C%3E)` yields `<>. The` url-decode` can receive multiple parameters (maximum 64). In this case, each parameter is decoded separately, and simply concatenated.

Available in AxoSyslog version 3.18 and later.

url-encode

Syntax: `$(url-encode ${MESSAGE})`

Description: The `url-encode` template function escapes strings for use as URI path or query parameter segments or form encoded HTTP POST data. All input characters that are not a-z, A-Z, 0-9, '-', '.', '_' or '~' are converted to their "URL escaped" version.

Available in AxoSyslog version 3.18 and later. (In version 3.16-3.17, this template function was called `urlencode`.)

uuid

Syntax: `$(uuid)`

Description: Generates a Universally Unique IDentifier (UUID) that complies with RFC4122 . That way, an UUID can be added to the message soon after it is received, so messages stored in multiple destinations can be identified. For example, when storing messages in a database and also in files, the UUID can be used to find a particular message both in the database and the files.

To generate a UUID, you can use a rewrite rule to create a new value-pair for the message.

Example: Using Universally Unique Identifiers

The following example adds a value-pair called `MESSAGE_UUID` to the message using a rewrite rule and a template.

```
rewrite r_add_uuid {
     set("$(uuid)" value("MESSAGE_UUID"));
 };

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

 log {
     source(s_network);
     rewrite(r_add_uuid);
     destination(d_file);
 };
```

> **Note**
>
> This template function is available only if the `tfuuid` module has been loaded.
>
> By default, AxoSyslog loads every available module. For details, see Loading modules.

13.1.8 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 AxoSyslog 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 AxoSyslog. A dedicated Redis destination is available in AxoSyslog version 3.5. For details, see redis: Store 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, AxoSyslog 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));
    };
```

13.2 Modify messages using rewrite rules

> ### FilterX or classic filters, parsers, and rewrite rules?
>
> **FilterX replaces filters, parsers, and rewrite rules with a single typed language that can also handle nested JSON and other complex data. You don't have to migrate: the classic blocks keep working, and you can use both in the same log path. To convert your existing configuration, see Update to FilterX.**

The AxoSyslog 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 AxoSyslog 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.conf` 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.**

13.2.1 Replace 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. You can also rewrite the structured-data fields of messages complying to the RFC5424 (IETF-syslog) message format.

> **Note**
>
> **Hard macros cannot be modified. For details, see Hard versus soft macros.**

See also the equivalent FilterX function, `regexp_subst()` , or `str_replace()` for literal strings.

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.

> **Note**
>
> **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"));};
```

Options

The `subst()` rewrite rule has the following options.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

flags()

Type:	list of flags
Default:	empty set

Description: The flags of the regular expression. The available flags depend on the `type()` of the regular expression. For case-insensitive searches, use `flags(ignore-case)`. To replace every occurrence of the pattern, use `flags(global)`. Note that the `store-matches` flag is automatically enabled in rewrite rules. For details, see Regular expressions.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

type()

Type:	`pcre` , `string` , `glob`
Default:	`pcre`

Description: Sets how AxoSyslog interprets the search pattern: as a Perl Compatible Regular Expression (`pcre` , used by default), a literal string search (`string`), or a glob pattern without regular expression support (`glob`). For details, see Options of regular expressions.

value()

Type:	name of a message field
Default:	`MESSAGE`

Description: Selects the field of the message that the rewrite rule modifies. If you don't set it, the rule operates on the `MESSAGE` field.

Write the name of the field without the `$` prefix, for example, `value("HOST")` , not `value("$HOST")` . The `$` prefix is only needed in templates, and AxoSyslog logs a warning if you use it here. You cannot set a hard macro in the `value()` option, AxoSyslog rejects the configuration with an error.

13.2.2 Set 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 type of the field. Where you can use of templates in `set()` and `groupset()`, you can use type-casting, and the type information is properly promoted. For details, see Specifying data types in value-pairs.

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)). Note that the rewrite operation completely replaces any previous value of that field.

> **Note**
>
> **Hard macros cannot be modified. For details, see Hard versus soft macros.**

See also the equivalent FilterX operation, assigning a value .

Use the following syntax:

Declaration

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

Example: Set 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 Create custom SDATA fields.

> **Note**
>
> **The `severity` and `facility` fields can only be set by the `set-severity()` rewrite functions. For more information, see Set the severity field.**

Options

The `set()` rewrite rule has the following options. In addition to `condition()`, `internal()`, and `value()`, `set()` accepts the template options listed below, which control how AxoSyslog expands the template in its first argument.

```
    rewrite <name_of_the_rule> {
         set("<string to include>", value(<field name>), on-error("fallback-to-
string"));
     };
```

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don’t match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

frac-digits()

Type:	number
Default:	`0`

Description: The number of digits that AxoSyslog stores when it expands a timestamp macro with fractions of a second. Overrides the global `frac-digits()` option for this rule.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

local-time-zone()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Sets the timezone that AxoSyslog uses when it expands a timestamp macro as a local time.

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `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()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Sets the timezone that AxoSyslog uses when it expands a timestamp macro. The `time-zone()` option is an alias of `send-time-zone()`.

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.

template-escape()

Type:	`yes`, `no`
Default:	`no`

Description: Turns on escaping for the `'`, `"`, and backspace characters in the expanded value.

time-zone()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Alias of `send-time-zone()`.

ts-format()

Type:	`rfc3164`, `bsd`, `rfc3339`, `iso`
Default:	`rfc3164`

Description: Overrides the global `ts-format()` option for the timestamp macros that this rule expands.

value()

Type:	name of a message field
Default:	`MESSAGE`

Description: Selects the field of the message that the rewrite rule modifies. If you don't set it, the rule operates on the `MESSAGE` field.

Write the name of the field without the `$` prefix, for example, `value("HOST")`, not `value("$HOST")`. The `$` prefix is only needed in templates, and AxoSyslog logs a warning if you use it here. You cannot set a hard macro in the `value()` option, AxoSyslog rejects the configuration with an error.

13.2.3 Set the severity field

It is possible to configure the `severity` field with the `set-severity()` rewrite function. When configured, the `set-severity()` rewrite function will only rewrite the `$SEVERITY` field in the message to the first parameter value specified in the function.

> **Note**
>
> **If the parameter value is not a valid parameter value, the function ignores it and sends a debug message, but the AxoSyslog application still sends the message.**

FilterX has no `set_severity()` counterpart. See the related `set_pri()` function, which sets the combined PRI value (`facility * 8 + severity`), so you cannot set the severity without also setting the facility. It also takes a number, not a severity name.

Declaration

```
rewrite <name_of_the_rule> {
    set-severity("severity string or number");
};
```

Parameters

The `set-severity()` rewrite function has a single, mandatory parameter that can be defined as follows:

```
set-severity( "parameter1" );
```

Accepted values

The `set-severity()` rewrite function accepts numeric values, named values, and aliases. Aliases are available in AxoSyslog version 4.6 and later.

Numerical Code	Named Value	Alias
0	`emerg`	`SYSLOG_SEVERITY_CODE(0)`

0	`emergency`	`SYSLOG_SEVERITY_CODE(0)`
0	`panic`	`SYSLOG_SEVERITY_CODE(0)`
1	`alert`	`SYSLOG_SEVERITY_CODE(1)`
2	`crit`	`SYSLOG_SEVERITY_CODE(2)`
2	`critical`	`SYSLOG_SEVERITY_CODE(2)`
2	`fatal`	`SYSLOG_SEVERITY_CODE(2)`
3	`err`	`SYSLOG_SEVERITY_CODE(3)`
3	`error`	`SYSLOG_SEVERITY_CODE(3)`
4	`warning`	`SYSLOG_SEVERITY_CODE(4)`
4	`warn`	`SYSLOG_SEVERITY_CODE(4)`
5	`notice`	`SYSLOG_SEVERITY_CODE(5)`
6	`info`	`SYSLOG_SEVERITY_CODE(6)`
6	`log`	`SYSLOG_SEVERITY_CODE(6)`
7	`debug`	`SYSLOG_SEVERITY_CODE(7)`

Example usage for the set-severity() rewrite function

The following examples use the `set-severity()` rewrite function.

Using a named value:

```
rewrite {
    set-severity("info");
};
```

Using a numeric value:

```
rewrite {
    set-severity("6");
};
```

Using a template:

```
rewrite {
    set-severity("${.json.severity}");
};
```

Options

The `set-severity()` rewrite rule has the following option.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

13.2.4 Set the facility field

It is possible to set the `facility` field with the `set-facility()` rewrite function. When set, the `set-facility()` rewrite function will only rewrite the `${FACILITY}` field in the message to the first parameter value specified in the function.

> **Note**
>
> **If the parameter value is not a valid parameter value, the function ignores it and sends a debug message, but the application still sends the message.**

FilterX has no `set_facility()` counterpart. See the related `set_pri()` function, which sets the combined PRI value (`facility * 8 + severity`), so you cannot set the facility without also setting the severity.

Declaration

```
log {
    source { system(); };
    if (program("postfix")) {
      rewrite { set-facility("mail"); };
    };
    destination { file("/var/log/mail.log"); };
    flags(flow-control);
};
```

Parameters

The `set-facility()` rewrite function has a single, mandatory parameter that can be defined as follows:

```
set-facility( "parameter1" );
```

Accepted values

The `set-facility()` rewrite function accepts the following numeric strings and named values:

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)

Example usage for the set-facility() rewrite function

The following two `set-facility()` examples are equivalent:

```
rewrite {
  set-facility("security");
  set-facility("13");
  };
```

Options

The `set-facility()` rewrite rule has the following option.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

13.2.5 Set the priority of a message

You can set the `PRI` value of a BSD or IETF syslog message with the `set-pri()` rewrite function by specifying a template string. This is useful, for example, if incoming messages do not have a `PRI` value specified by default, but a PRI value is required for filtering purposes.

When configured, the `set-pri()` function will only rewrite the `PRI` value of the message field.

> **Note**
>
> **If the specified parameter value is not a valid value, the function ignores it and sends a debug message. However, the AxoSyslog application will still send the message.**

See also the equivalent FilterX function, `set_pri()` , which takes an integer instead of a template string.

Declaration

```
rewrite <rule-name> {
   set-pri("template-string");
};
```

Parameters

The `set-pri()` rewrite function expects a template string as its only parameter, for example:

- `set-pri("42");`
- `set-pri("$.json.priority");`

Accepted values

The template string specified for the `set-pri()` rewrite function must expand to a natural number in the interval of 0–1023, inclusive. This means that if you, for example, extract the value from a syslog `<PRI>` header (such as `<42>`), then you need to remove

the opening and closing brackets (`< >`) in the specified template string.

Example: Temporarily raising the priority of an application

In the following example, the `set-pri()` rewrite function is used to temporarily raise the priority of the application `myprogram` :

```
log {
   source { system(); };
   if (program("myprogram")){
   rewrite { set-pri("92"); };
   };
   destination { file("/var/log/mail.log"); };
   flags(flow-control);
}
```

Example: Change the priority in JSON-formatted log

In the following example, an application sends log messages in the following JSON format:

```
{
   "time": "2003-10-11T22:14:15.003Z",
   "host": "mymachine",
   "priority": "165",
   "message": "An application event log entry."
}
```

You can parse these logs with the JSON parser function:

```
{
   parser p_json {
   json-parser (prefix(".json."));
}
```

As the application message contains a valid priority field, you can use the `set-pri()` rewrite function to modify the priority of the message:

```
set-pri("$.json.priority");
```

Options

The `set-pri()` rewrite rule has the following option.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

13.2.6 Set match variables

Match macros (`$1, $2, ... $255`) are temporary variables. You can use them for general purposes when operating with list-like items. For example, the match() filter stores capture group results in match variables when the `store-matches` flag is set, or the JSON parser produces match variables if the parsed JSON data is an array.

It is possible to set match variables in a single operation with the `set-matches()` rewrite function. `set-matches()` uses AxoSyslog list expressions to set `$1, $2, ... $255` , so it can be considered as a conversion function between AxoSyslog lists and match variables.

> **Note**
>
> **To convert match variables into a AxoSyslog list, use the `$*` macro, which can be further manipulated using list template functions , or turned into a list in type-aware destinations.**

> **Note**
>
> **To reset match variables to be empty, use the `unset-matches()` rewrite rule.**

Declaration

```
rewrite <name_of_the_rule> {
    set-matches("<list-expression or list-based template function>");
};
```

Example for the set-matches() rewrite function

In the following two examples, `$1` , `$2` , and `$3` will be set to `foo` , `bar` , and `baz` , respectively.

Using strings:

```
rewrite {
    set-matches("foo,bar,baz");
};
```

Using a list template function:

```
rewrite {
    set-matches("$(explode ':' 'foo:bar:baz')");
};
```

unset-matches()

The `unset-matches()` rewrite rule clears every match variable, that is, it resets `$1, $2, ... $255` to empty. Use it when you want to make sure that match variables left over from an earlier filter, parser, or rewrite rule do not leak into later parts of the log path.

```
rewrite <name_of_the_rule> {
    unset-matches();
};
```

Example for the unset-matches() rewrite function

The following log path sets three match variables, then clears them again, so `$1`, `$2`, and `$3` are empty when the message reaches the destination.

```
log {
    source(s_local);
    rewrite { set-matches("foo,bar,baz"); };
    rewrite { unset-matches(); };
    destination(d_local);
};
```

Options

Both `set-matches()` and `unset-matches()` have the following option.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

13.2.7 Unset 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)). Note that the unset operation completely deletes any previous value of the field that you apply it on.

> **Note**
>
> **Hard macros cannot be modified. For details, see Hard versus soft macros.**

Unset a field

Use the following syntax:

```
rewrite <name_of_the_rule> {
    unset(value("<field-name>"));
};
```

Example: Unset a message field

The following example unsets the HOST field of the message.

```
rewrite r_rewrite_unset{
    unset(value("HOST"));
};
```

If you're using FilterX, see also the equivalent `unset()` function.

Unset group

To unset a group of fields, you can use the `groupunset()` rewrite rule.

```
rewrite <name_of_the_rule> {
    groupunset(values("<expression-for-field-names>"));
};
```

Example: Unset a group of fields

The following rule clears all SDATA fields:

```
rewrite r_rewrite_unset_SDATA{
    groupunset(values(".SDATA.*"));
};
```

Options

The `unset()` rewrite rule has the following options.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

value()

Type:	name of a message field
Default:	`MESSAGE`

Description: Selects the field of the message that the rewrite rule modifies. If you don't set it, the rule operates on the `MESSAGE` field.

Write the name of the field without the `$` prefix, for example, `value("HOST")`, not `value("$HOST")`. The `$` prefix is only needed in templates, and AxoSyslog logs a warning if you use it here. You cannot set a hard macro in the `value()` option, AxoSyslog rejects the configuration with an error.

The `groupunset()` rewrite rule has the `condition()` option described above, and the following option.

values()

Type:	list of field names or globs
Default:	N/A

Description: Specifies the fields to unset. You can list the fields explicitly as a space-separated list of double-quoted names, or select multiple fields with glob expressions, for example, `values(".SDATA.*")`. This option is mandatory.

13.2.8 Rename message fields

If you want to change the name of a field of a message, you can use `rename()` rewrite rules. This can be also achieved via using `set()` and `unset()` but those require extra conditions and two operation instead of one.

The `rename()` rewrite rule uses positional arguments and they are both required. It supports condition rewrite. For more information, see Conditional rewrites.

See also the related FilterX function `move()` . Note that `rename()` does nothing if the source field is missing, while `move()` fails in that case.

Declaration

```
rewrite r_rewrite_rename {
    rename("<string1>" "<string2>");
};
```

Example usage for the rename() rewrite function

The following example renames the `.app.name` into `.container` if the `.app.name` exists. Otherwise, it does nothing.

```
rewrite r_rewrite_rename {
    rename(".app.name" ".container");
};
```

Options

The `rename()` rewrite rule has the following option.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

13.2.9 Create 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 Axoflow, the developer of AxoSyslog.)

Example: Rewriting custom SDATA fields

The following example sets the sequence ID field of the RFC5424-formatted (IETF-syslog) messages to a fixed value. This field is a predefined SDATA field with a reserved SD-ID, therefore its name does not contain the `@` character.

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

It is also possible to set the value of a field that does not exist yet, and create a new, custom name-value pair that is associated with the message. The following example creates the `.SDATA.groupID.fieldID@18372.4` field and sets its value to `yes`. If you use the `${.SDATA.groupID.fieldID@18372.4}` macro in a template or SQL table, its value will be `yes` for every message that was processed with this rewrite rule, and empty for every other message.

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.

In FilterX, you can set the same fields with an assignment, for example, `${.SDATA.meta.sequenceId} = 55555;`. To read SDATA fields, see the related Handle SDATA in RFC5424 log records functions.

13.2.10 Set 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.
- You can set the type of the field. Where you can use of templates in `set()` and `groupset()` , you can use type-casting, and the type information is properly promoted. For details, see Specifying data types in value-pairs.

See also the related FilterX function `set_fields()` , which sets multiple keys of a dict. It differs from `groupset()` : it selects keys explicitly instead of by glob expression, has no equivalent of the `$_` macro, and it can also create new keys.

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.*"))
```

Options

The `groupset()` rewrite rule has the following options.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

values()

Type:	list of field names or globs
Default:	N/A

Description: Specifies the fields to modify. You can list the fields explicitly as a space-separated list of double-quoted names, or select multiple fields with glob expressions, for example, `values(".USER.*")`. This option is mandatory. Note that `groupset()` only modifies existing fields, it doesn't create new ones.

13.2.11 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. Using `map-value-pairs()` retains type data if available.

Available in AxoSyslog version 3.10 and later.

FilterX has no bulk equivalent. To rename fields one by one, see the related `move()` function.

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")))
    );
};
```

Options

The `map-value-pairs()` parser accepts the `value-pairs()` options listed below. Note that `map-value-pairs()` takes these options directly, without wrapping them into a `value-pairs()` block.

cast()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If set to `yes` , AxoSyslog converts every mapped value to string, discarding the type information of the name-value pairs. By default, `map-value-pairs()` retains the type of the values.

exclude()

Description: Excludes the name-value pairs matching the specified glob patterns from the mapping. For details, see `exclude()` .

include-bytes()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If set to `yes` , AxoSyslog also maps the name-value pairs that have the `bytes` or `protobuf` type. By default, these are skipped.

key()

Description: Selects the name-value pairs to map, using glob patterns. You can combine it with `rekey()` to rename the selected name-value pairs. For details, see `key()` .

pair()

Description: Adds a new name-value pair with the specified name and value. The value can be a template or a template function. For details, see `pair()` .

rekey()

Description: Renames the selected name-value pairs, for example, by adding or replacing a prefix, changing the case of the names, or removing levels from the names. For details, see `rekey()` .

scope()

Description: Selects a predefined group of name-value pairs to map, for example, `nv-pairs` or `all-nv-pairs` . For details, see `scope()` .

13.2.12 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.

> **Note**
>
> **Using conditions in rewrite rules can simplify your AxoSyslog configuration file, as you do not need to create separate log paths to modify certain messages.**

See also the equivalent FilterX construct, conditional statements . Use an `if` statement to match the behavior of `condition()` : a bare comparison statement drops the messages that do not match it, while `condition()` passes them on unmodified.

Using conditional rewrite

The following procedure summarizes how conditional rewrite rules (rewrite rules that have the `condition()` parameter set) work. The following configuration snippet is used to illustrate the procedure:

```
rewrite r_rewrite_set{
     set(
         "myhost",
         value("HOST")
         condition(program("myapplication"))
     );
 };
 log {
     source(s1);
     rewrite(r_rewrite_set);
     destination(d1);
 };
```

1. The log path receives a message from the source (`s1`).

2. The rewrite rule (`r_rewrite_set`) evaluates the condition. If the message matches the condition (the `PROGRAM` field of the message is `myapplication`), AxoSyslog 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)));};
```

13.2.13 Add and delete 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>");
};
```

Templates (macros, template functions) can be used when specifying tags, for example, `set-tag("dyn::$HOST");` .

Options

The `set-tag()` and `clear-tag()` rewrite rules have the following option.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

13.2.14 Rewrite the timezone of a message

Starting with version 3.24 of the AxoSyslog application, you can manipulate the timezone information of messages using rewrite rules. You can:

- Set a timezone to a specific value
- Fix a timezone if it was improperly parsed
- Assuming the sender is sending messages in near-real-time, AxoSyslog can guess the timezone

By default, these operations modify the date-related macros of the message that correspond to the date the message was sent (that is, the `S_` macros). You can modify the dates when AxoSyslog has received the messages (that is, the R_ macros), but this is rarely needed. To do so, include the `time-stamp(recvd)` option in the operation, for example:

```
rewrite { fix-time-zone("EST5EDT" time-stamp(recvd)); };
```

If you're using FilterX, see also the equivalent `set_timezone()` function.

fix-time-zone()

Use the `fix-time-zone()` operation to correct the timezone of a message if it was parsed incorrectly for some reason, or if the client did not include any timezone information in the message. You can specify the new timezone as the name of a timezone, or as a template string. For example, use the following rewrite rule to set the timezone to EST5EDT:

```
rewrite { fix-time-zone("EST5EDT"); };
```

If you have lots of clients that do not send timezone information in the log messages, you can create a database file that stores the timezone of the clients, and feed this data to AxoSyslog using the `add-contextual-data()` feature. For details, see Adding metadata from an external file.

See also the equivalent FilterX function, `fix_timezone()` .

guess-time-zone()

Use the `guess-time-zone()` operation attempts to set the timezone of the message automatically, using heuristics on the timestamps. Normally the AxoSyslog application performs this operation automatically when it parses the incoming message. Using this operation in a rewrite rule can be useful if you cannot parse the incoming message for some reason (and use the `flags(no-parse)` option in your source, but you want to set the timezone automatically later (for example, after you have preprocessed the message).

Using this operation is identical to using the `flags(guess-timezone)` flag in the source.

See also the equivalent FilterX function, `guess_timezone()` .

set-time-zone()

Use the `set-time-zone()` operation to set the timezone of the message to a specific value, that is to convert an existing timezone to a different one. This operation is identical to setting the `time-zone()` option in a destination or as a global option, but can be applied selectively to the messages using conditions.

Options

The `fix-time-zone()` , `guess-time-zone()` , and `set-time-zone()` rewrite rules have the following options.

condition()

Type:	filter expression
Default:	N/A

Description: Applies the rewrite rule only to the messages that match the specified filter expression. Messages that don't match the filter pass through the rule unmodified, and continue to the next element of the log path. You can use any filter expression here, and you can reference an existing filter with the `filter()` function. For details, see Conditional rewrites.

time-stamp()

Type:	`stamp` , `recvd`
Default:	`stamp`

Description: Selects the timestamp that the rule modifies. With `time-stamp(stamp)` , the rule changes the date the message was sent, that is, the `S_` macros. With `time-stamp(recvd)` , it changes the date AxoSyslog received the message, that is, the `R_` macros.

13.2.15 Anonymize 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 AxoSyslog 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, AxoSyslog replaces the number `5542043004559005` with

`554204******9005`.

13.3 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.

> **Note**
>
> **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));
};
```

> **Note**
>
> **Adding the `flags(ignore-case)` option to glob patterns does not disable case sensitivity.**

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.

Example: matching / non-matching patterns for the input string 'exam'

For the input string `'exam'` ,

- the following patterns will match:
  - `'ex'` (the pattern contains the initial characters of the input string in the exact same order)
  - `'exam'` (the pattern is an exact match for the input string)
- the following patterns will not match:
  - `'example'` (the pattern is longer than the input string)
  - `'hexameter'` (the pattern's initial characters do not match the input string's characters in the exact same order, and the pattern is longer than the input string)

13.3.1 Options of regular expressions

This chapter lists regular expressions supported by AxoSyslog and their available supported `type()` and `flags()` options.

By default, AxoSyslog uses PCRE-style regular expressions. To use other expression types, add the `type()` option after the regular expression.

The AxoSyslog application supports the following regular expression `type()` options:

- Perl Compatible Regular Expressions (`pcre`)
- Literal string searches (`string`)
- Glob patterns without regular expression support (`glob`)

By default, AxoSyslog uses PCRE-style regular expressions, which are supported on every platform starting with AxoSyslog version 3.1. To use other expression types, add the `type()` option after the regular expression.

Perl Compatible Regular Expressions (pcre)

Description: Uses Perl Compatible Regular Expressions (PCRE). If the `type()` parameter is not specified, AxoSyslog uses PCRE regular expressions by default.

PCRE flags

PCRE-style regular expressions have the following `flags()` options:

- `disable-jit` : Switches off the just-in-time compilation function for PCRE regular expressions .
- `dupnames` : Allows using duplicate names for named subpatterns . Configuration example:

```
filter { match("(?<DN>foo)|(?<DN>bar)" value(MSG) flags(store-matches,
dupnames)); };
...
destination { file(/dev/stdout template("$DN\n")); };
```

- `global` : Usable only in rewrite rules, `flags("global")` matches for every occurrence of the expression, not only the first one.
- `ignore-case` : Disables case-sensitivity.

- `newline`:
  When configured, it changes the newline definition used in PCRE regular expressions to accept either of the following:
  - a single carriage-return
  - linefeed
  - the sequence carriage-return and linefeed (`\\r` , `\\n` and `\\r\\n` , respectively)

  This newline definition is used when the circumflex and dollar patterns (`^` and `$`) are matched against an input. By default, PCRE interprets the linefeed character as indicating the end of a line. It does not affect the `\\r` , `\\n` or `\\R` characters used in patterns.

- `store-matches` : Stores 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.

  > **Note**
  >
  > **To convert match variables into a AxoSyslog list, use the `$*` macro, which can be further manipulated using List manipulation , or turned into a list in type-aware destinations.**

- `unicode`:
  Use Unicode support for UTF-8 matches: UTF-8 character sequences are handled as single characters.

- `utf8` : An alias for the `unicode` flag.

Literal string searches (string)

Description: Matches 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.

String search flags

Literal string searches have the following `flags()` options:

- `global` : Usable only in rewrite rules, `flags("global")` matches for every occurrence of the expression, not only the first one.

- `ignore-case` : Disables case-sensitivity.
- `prefix` : During the matching process, patterns (also called search expressions) are matched against the input string starting from the beginning of the input string, and the input string is matched only for the maximum character length of the pattern. The initial characters of the pattern and the input string must be identical in the exact same order, and the pattern's length is definitive for the matching process (that is, if the pattern is longer than the input string, the match will fail).
- `store-matches` : Stores 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.

  > **Note**
  >
  > **To convert match variables into a AxoSyslog list, use the `$*` macro, which can be further manipulated using List manipulation , or turned into a list in type-aware destinations.**

- `substring` : The given literal string will match when the pattern is found within the input. Unlike `flags("prefix")` , the pattern does not have to be identical with the given literal string.

Glob patterns without regular expression support (glob)

Description: Matches the strings against a pattern containing `*` and `?` wildcards, without regular expression and character range support. The advantage of glob patterns to regular expressions is that globs can be processed much faster.

- `*` : matches an arbitrary string, including an empty string
- `?` : matches an arbitrary character
- The wildcards can match the `/` character.
- You cannot use the `*` and `?` literally in the pattern.

Glob patterns don't support any flags.

13.3.2 Optimizing regular expressions

The `host()`, `match()`, and `program()` filter functions and some other 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

Extract name-value pairs from message content with parser drivers for JSON, XML, CSV, key=value, regexp, vendor-specific formats, and pattern databases.

> ### FilterX or classic filters, parsers, and rewrite rules?
>
> **FilterX replaces filters, parsers, and rewrite rules with a single typed language that can also handle nested JSON and other complex data. You don't have to migrate: the classic blocks keep working, and you can use both in the same log path. To convert your existing configuration, see Update to FilterX.**

The filters and default macros of AxoSyslog 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 AxoSyslog application provides the following possibilities to parse the messages, or parts of the messages, as shown on the following list. There are several built-in parsers for application-specific logs.

Note that by default, AxoSyslog parses every message as a syslog message. To disable parsing the message as a syslog message, 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.

14.1 Apache access log parser

The Apache access log parser can parse the access log messages of the Apache HTTP Server. The AxoSyslog application can separate these log messages to name-value pairs. For details on using value-pairs in AxoSyslog 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
```

Starting with version 3.21, virtualhost and the port of the virtualhost (vhost) is also supported, for example:

```
foo.com:443 1.2.3.4 - - [15/Apr/2019:14:30:16 -0400] "GET /bar.html HTTP/2.0"
500 - "https://foo.com/referer.html" "Mozilla/5.0 ..."
```

FilterX has no Apache parser, but you can parse these logs with `parse_csv()` . For a related example, see Parse Apache log files .

The AxoSyslog application extracts every field into name-value pairs, and adds the `.apache.` prefix to the name of the field.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `apache-accesslog-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/apache/apache.conf on GitHub.

Declaration:

```
parser parser_name {
     apache-accesslog-parser(
         prefix()
     );
 };
```

The parser extracts the following fields from the messages: `vhost`, `port`, `clientip`, `ident`, `auth`, `timestamp`, `rawrequest`, `response`, `bytes`, `referrer`, and `agent`. The `rawrequest` field is further segmented into the `verb`, `request`, and `httpversion` fields. The AxoSyslog `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 the `.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);
 };
```

14.1.1 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `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}`).

14.2 Check Point Log Exporter parser

The Check Point Log Exporter parser can parse Check Point log messages. These messages do not completely comply with the syslog RFCs, making them difficult to parse. The `checkpoint-parser()` of AxoSyslog solves this problem, and can separate these log messages to name-value pairs. For details on using value-pairs in AxoSyslog see Structuring macros, metadata, and other value-pairs. The parser can parse messages in the following formats:

```
<PRI><VERSION> <YYYY-MM-DD> <HH-MM-SS> <PROGRAM> <PID> <MSGID> - [key1:value1;
key2:value2; ... ]
```

For example:

```
<134>1 2018-03-21 17:25:25 MDS-72 CheckPoint 13752 - [action:"Update";
flags:"150784"; ifdir:"inbound"; logid:"160571424"; loguid:"
{0x5ab27965,0x0,0x5b20a8c0,0x7d5707b6}";]
```

Splunk format:

```
time=1557767758|hostname=r80test|product=Firewall|layer_name=Network|layer_uuid=c02
64a80-1832-4fce-8a90-
d0849dc4ba33|match_id=1|parent_rule=0|rule_action=Accept|rule_uid=4420bdc0-19f3-
4a3e-8954-03b742cd3aee|action=Accept|ifdir=inbound|ifname=eth0|logid=0|loguid=
{0x5cd9a64e,0x0,0x5060a8c0,0xc0000001}|origin=192.168.96.80|originsicname=cn\=cp_mg
mt,o\=r80test..ymydp2|sequencenum=1|time=1557767758|version=5|dst=192.168.96.80|inz
one=Internal|outzone=Local|proto=6|s_port=63945|service=443|service_id=https|src=19
2.168.96.27|
```

If you find a message that the `checkpoint-parser()` cannot properly parse, contact us , so we can improve the parser.

By default, the Check Point-specific fields are extracted into name-value pairs prefixed with `.checkpoint`. For example, the `action` in the previous message becomes `${.checkpoint.action}`. You can change the prefix using the `prefix` option of the parser.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `checkpoint-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/checkpoint/plugin.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { network(flags(no-parse)); };
     parser { checkpoint-parser(); };
     destination { ... };
 };
```

Note that the parser expects that the entire incorrectly formatted syslog message (starting with its `<PRI>` value) is in `$MSG` , which you can achieve by using `flags(no-parse)` on the input driver.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `checkpoint-parser()` uses the `.checkpoint.` prefix. To modify it, use the following format:

```
parser {
     checkpoint-parser(prefix("myprefix."));
 };
```

14.3 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 AxoSyslog 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 AxoSyslog 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
<189>32: foo: *Apr 29 13:58:46.411: %SYSMGR-STANDBY-3-SHUTDOWN_START: The
System Manager has started the shutdown procedure.
```

> **Note**
>
> **Not every Cisco log message conforms to this format. If you find a message that the `cisco-parser()` cannot properly parse, contact us , so we can improve the parser.**

The AxoSyslog 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.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `cisco-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/cisco/plugin.conf on GitHub.

Declaration:

```
@version: 4.29
@include "scl.conf"
log {
    source { network(
                transport("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.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `cisco-parser()` uses the `.cisco.` prefix. To modify it, use the following format:

```
parser {
     cisco-parser(prefix("myprefix."));
 };
```

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}`).

14.4 Parsing messages with comma-separated and similar values

The AxoSyslog application can separate parts of log messages (that is, the contents of the `${MESSAGE}` macro) at delimiter characters or strings to named fields (columns) using the csv (comma-separated-values) parser 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 AxoSyslog 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

Starting with AxoSyslog version 4.5, you can omit the `columns()` option, and extract the values into matches (`$1` , `$2` , `$3` , and so on), which are available as the anonymous list `$*` . For example:

```
@version: current

log {
    source { tcp(port(2000) flags(no-parse)); };

    parser { csv-parser(delimiters(',') dialect(escape-backslash)); };
    destination { stdout(template("$ISODATE $*\n")); };
};
```

See also the equivalent FilterX function, `parse_csv()` . Note that it uses the `delimiter` option instead of `delimiters()` .

Example: Segmenting hostnames separated with a dash

The following example separates hostnames like `example-1` and `example-2` into two parts.

```
   parser p_hostname_segmentation {
        csv-parser(columns("HOSTNAME.NAME", "HOSTNAME.ID")
        delimiters("-")
        flags(escape-none)
        template("${HOST}"));
    };
    destination d_file {
        file("/var/log/messages-${HOSTNAME.NAME:-examplehost}");
    };
    log {
        source(s_local);
        parser(p_hostname_segmentation);
        destination(d_file);
    };
```

Example: Parsing Apache log files

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

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

Here is a sample message:

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

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

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

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

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

Example: Segmenting a part of a message

Multiple parsers can be used to split a part of an already parsed message into further segments. The following example splits the timestamp of a parsed Apache log message into separate fields.

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

Further examples:

- For an example on using the `greedy` option, see Example: Adding the end of the message to the last column .

14.4.1 Options of CSV parsers

The AxoSyslog application can separate parts of log messages (that is, the contents of the `${MESSAGE}` macro) at delimiter characters or strings to named fields (columns) using the csv (comma-separated-values) parser 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 AxoSyslog 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

columns()

Synopsis: columns(“PARSER.COLUMN1”, “PARSER.COLUMN2”, ...)

Description: Specifies the name of the columns to separate messages to. These names will be automatically available as macros. The values of these macros do not include the delimiters.

Starting with AxoSyslog version 4.5, you can omit the `columns()` option, and extract the values into matches (`$1` , `$2` , `$3` , and so on), which are available as the anonymous list `$*` . For example:

```
@version: current

log {
    source { tcp(port(2000) flags(no-parse)); };

    parser { csv-parser(delimiters(',') dialect(escape-backslash)); };
    destination { stdout(template("$ISODATE $*\n")); };
};
```

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, AxoSyslog uses space as a delimiter. If you want to use only the strings as delimiters, you have to disable the space delimiter, for example: `delimiters(chars(""), strings("<delimiter_string>"))`

Otherwise, AxoSyslog will use the string delimiters in addition to the default character delimiter, so `delimiters(strings("=="))` actually equals `delimiters(chars(" "), strings("=="))`, and not `delimiters(chars(""), strings("=="))`

Multiple delimiters

If you use more than one delimiter, note the following points:

- AxoSyslog 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 may 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 input, AxoSyslog uses the string delimiter.

dialect()

Synopsis:	`escape-none`, `escape-backslash`, `escape-double-char`, or `escape-backslash-with-sequences`

Description: Specifies how to handle escaping in the parsed message. Default value: `escape-none`

```
parser p_demo_parser {
    csv-parser(
        prefix(".csv.")
        delimiters(" ")
        dialect(escape-backslash)
        flags(strip-whitespace, greedy)
        columns("column1", "column2", "column3")
    );
};
```

The following values are available.

- `escape-backslash` : The parsed message uses the backslash (`\\`) character to escape quote characters.
- `escape-backslash-with-sequences` : The parsed message uses `""` as an escape character but also supports C-style escape sequences, like `\n` or `\r` . Available in AxoSyslog version 4.0 and later.
- `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.

drop-invalid()

Accepted values:	`yes` , `no`
Default:	`no`

Description:

When enabled, the parser doesn't process messages that don't match the parser. For example, a message that has fewer columns than specified, or more columns when the `greedy` flag isn't enabled. Using `drop-invalid(yes)` practically turns the parser into a special filter that matches messages with the predefined number of columns (using the specified delimiters). `drop-invalid(yes)` is equivalent to setting the `drop-invalid` value in `flags()` .

> **Note**
>
> **Messages dropped as invalid can be processed by a `fallback` log path. For details on the `fallback` option, see Log path flags.**

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 enabled, the parser doesn't process messages that don't match the parser. For example, a message that has fewer columns than specified, or more columns when the `greedy` flag isn't enabled. Using `drop-invalid(yes)` practically turns the parser into a special filter that matches messages with the predefined number of columns (using the specified delimiters). `drop-invalid(yes)` is equivalent to setting the `drop-invalid` value in `flags()` .

  > **Note**
  >
  > **Messages dropped as invalid can be processed by a `fallback` log path. For details on the `fallback` option, see Log path flags.**

- *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.

- *strip-whitespace*: The `strip-whitespace` flag removes leading and trailing whitespaces from all columns.

Example: Adding the end of the message to the last column

If the `greedy` option is enabled, AxoSyslog 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));
```

internal()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

null()

Synopsis:	string

Description: If the value of a column is the value of the `null()` parameter, AxoSyslog 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

This parser does not have a default prefix. To configure a custom prefix, use the following format:

```
parser {
    csv-parser(prefix("myprefix."));
};
```

on-type-error()

Synopsis: string

Description: Specifies what to do when casting a parsed value to a specific data type fails. Note that the `flags(drop-invalid)` option and the `on-error()` global option also affects the behavior.

Accepts the same values as the `on-error()` global option .

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 .

quotes()

Synopsis: `quotes(<characters>)`

Description: Specifies a set of single-character quote symbols where the opening and closing characters are identical. Each character in the string is treated as a quote, and delimiter characters or strings enclosed between matching quote characters are ignored. For example, `quotes('"\'')` treats both `"` and `'` as quote characters. To define asymmetric opening and closing quote characters (such as `[` and `]`), use `quote-pairs()` instead.

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 .

14.5 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**
>
> **Note that parsing will fail if the format string does not match the entire template or field. Since by default AxoSyslog 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.**

In FilterX, this takes two related functions: `strptime()` creates a `datetime` value from a string, then `set_timestamp()` writes it to the message. Use its `stamp` argument for the equivalent of the `time-stamp(recvd)` option.

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, AxoSyslog 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, AxoSyslog will include the entire original message (including the original timestamp) in the ${MESSAGE} macro.

```
   source s_file {
        file("/tmp/input" flags(no-parse));
    };

    destination d_file {
        file(
            "/tmp/output"
            template("${S_ISODATE} ${MESSAGE}\n")
        );
    };

    log {
        source(s_file);
        parser { date-parser(format("%d/%b/%Y:%H:%M:%S %Z")
template("${MY_DATE}")); };
        destination(d_file);
    };
```

In the template option, you can use template functions to specify which part of the message to parse with the format string. The following example selects the first 24 characters of the ${MESSAGE} macro.

```
   date-parser(format("%d/%b/%Y:%H:%M:%S %Z") template("$(substr ${MESSAGE} 0 24)")
);
```

In AxoSyslog version 3.23 and later, you can specify a comma-separated list of formats to parse multiple date formats with a single parser. For example:

```
   date-parser(format(
        "%FT%T.%f",
        "%F %T,%f",
        "%F %T"
    ));
```

If you need to modify or correct the timezone of the message after parsing, see Rewrite the timezone of a message.

14.5.1 Options of date-parser() parsers

The `date-parser()` parser has the following options.

format()

Synopsis:	format(string)
Default:	

Description: Specifies the format how AxoSyslog should parse the date. You can use the following format elements:

- `%a` : The day of week, using the locale's weekday names. Either the abbreviated or full name may be specified.
- `%A` : Same as % `a` .
- `%b` : The month, using the locale's month names. Either the abbreviated or full name may be specified.
- `%B` : The same as `%b` .
- `%c` : The date and time, using the locale's date and time format.
- `%C` : The century number [0,99]. Leading zeros are permitted but not required. This conversion should be used in conjunction with the %y conversion.
- `%d` : The day of month [1,31]. Leading zeros are permitted but not required.
- `%D` : The date as `%m/%d/%y` .
- `%e` : The same as `%d` .
- `%F` : The date as `%Y-%m-%d` (the ISO 8601 date format).
- `%g` : The year corresponding to the ISO week number, without the century. (A NetBSD extension.)
- `%G` : The year corresponding to the ISO week number, with the century. (A NetBSD extension.)
- `%h` : The same as `%b` .
- `%H` : The hour (24-hour clock) [0,23]. Leading zeros are permitted but not required.
- `%I` : The hour (12-hour clock) [1,12]. Leading zeros are permitted but not required.
- `%j` : The day number of the year [1,366]. Leading zeros are permitted but not required.
- `%k` : The same as `%H` .
- `%l` : The same as `%I` .
- `%m` : The month number [1,12]. Leading zeros are permitted but not required.

- `%M` : The minute [0,59]. Leading zeros are permitted but not required.
- `%n` : Any white-space, including none.
- `%p` : The locale's equivalent of a.m. or p.m.
- `%r` : The time (12-hour clock) with `%p` , using the locale's time format.
- `%R` : The time as `%H:%M` .
- `%S` : The seconds [0,60]. Leading zeros are permitted but not required.
- `%s` : The number of seconds since the Epoch, UTC (see mktime(3)). (A NetBSD extension.)
- `%f` : Fraction of the second (with or without a leading dot).
- `%t` : Any white-space, including none.
- `%T` : The time as `%H:%M:%S` .
- `%u` : The day of the week as a decimal number, where Monday = 1. (A NetBSD extension.)
- `%U` : The week number of the year (Sunday as the first day of the week) as a decimal number [0,53]. Leading zeros are permitted but not required. All days in a year preceding the first Sunday are considered to be in week 0.
- `%V` : The ISO 8601:1988 week number as a decimal number. If the week (starting on Monday) that contains January 1 has more than three days in the new year, then it is considered the first week of the year. If it has fewer than four days in the new year, then it is considered the last week of the previous year. Weeks are numbered from 1 to 53. (A NetBSD extension.)
- `%w` : The weekday as a decimal number [0,6], with 0 representing Sunday. Leading zeros are permitted but not required.
- `%W` : The week number of the year (Monday as the first day of the week) as a decimal number [0,53]. Leading zeros are permitted but not required. All days in a year preceding the first Monday are considered to be in week 0.
- `%x` : The date, using the locale's date format.
- `%X` : The time, using the locale's time format.
- `%y` : The year within the 20th century [69,99] or the 21st century [0,68]. Leading zeros are permitted but not required. If specified in conjunction with `%C` , specifies the year [0,99] within that century.
- `%Y` : The year, including the century (i.e., 1996).
- `%Z` : Timezone in ascii format (for example, PST), or in format -/+0000, accepts `:` in the middle of timezones (ISO 8601)
- `%z` : Timezone in ascii format (for example, PST), or in format -/+0000, accepts `:` in the middle of timezones (ISO 8601) (Required element)
- `%%` : matches a literal `%` . No argument is converted.

Warning

When using the `%z` and `%Z` format codes, consider that while `%z` strictly expects a specified timezone, and triggers a warning if the timezone is missing, `%Z` does not trigger a warning if the timezone is not specified.

For further information about the `%z` and `%Z` format codes, see the 'DESCRIPTION' section on the srtptime(3) - NetBSD Manual Pages .

For example, for the date `01/Jan/2016:13:05:05 PST` use the following format string: `"%d/%b/%Y:%H:%M:%S %Z"`

In AxoSyslog version 3.23 and later, you can specify a comma-separated list of formats to parse multiple date formats with a single parser. For example:

```
date-parser(format(
     "%FT%T.%f",
     "%F %T,%f",
     "%F %T"
 ));
```

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}`).

flags()

Type:	guess-timezone
Default:	empty string

guess-timezone: Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp. For example:

```
date-parser(flags(guess-timezone));
```

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

time-stamp()

Synopsis:	`stamp` or `recvd`
Default:	stamp

Description: Determines if the parsed date values are treated as sent or received date. If you use `time-stamp(stamp)` , AxoSyslog adds the parsed date to the S_ macros (corresponding to the sent date). If you use `time-stamp(recvd)` , AxoSyslog 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, AxoSyslog 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.

value()

Synopsis:	string
Default:	

Available in AxoSyslog 4.1 and later.

Description: Instruct `date-parser()` to store the resulting timestamp in a name-value pair specified in `value()`, instead of changing the timestamp value of the LogMessage.

14.6 db-parser: Process message content with a pattern database (patterndb)

14.6.1 Classifying log messages

The AxoSyslog application can compare the contents of the received log messages to predefined message patterns. By comparing the messages to the known patterns, AxoSyslog 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, AxoSyslog uses a method called longest prefix match radix tree. This means that AxoSyslog 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, AxoSyslog 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 AxoSyslog 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 AxoSyslog 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`

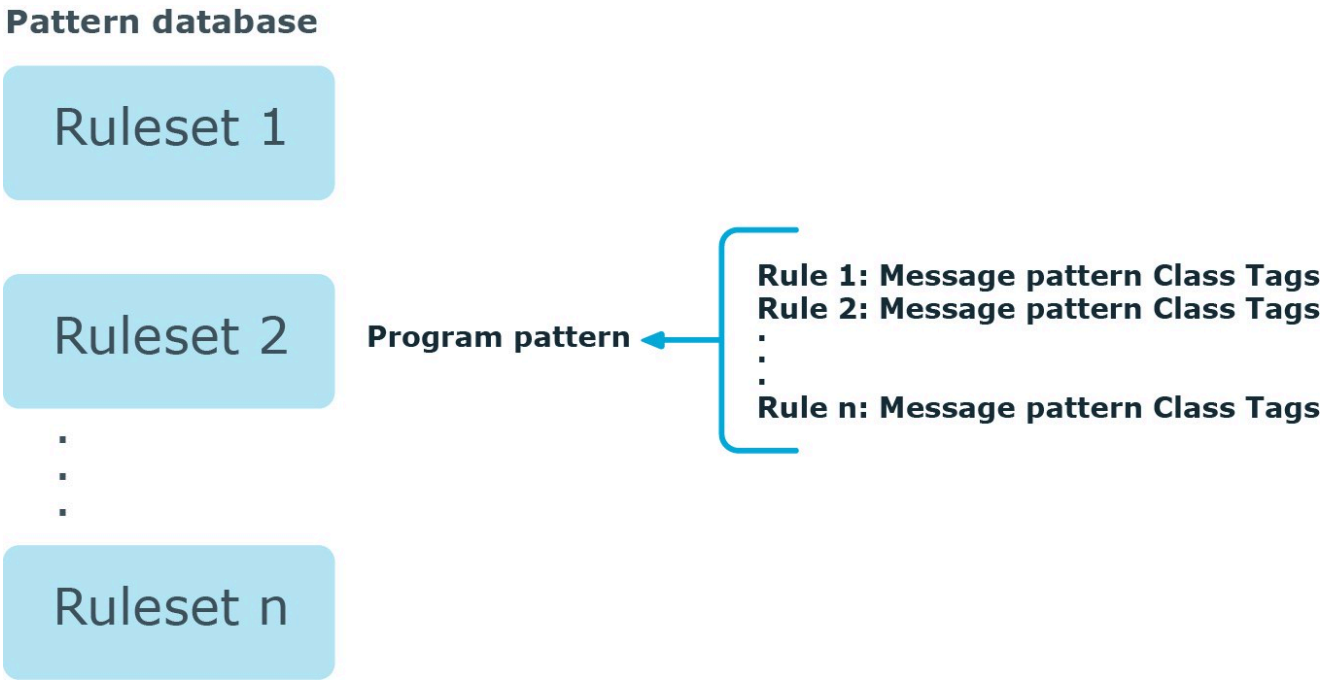
A regular expression matching this message from the logcheck project: `Accepted (gssapi(-with-mic|-keyex)?|rsa|dsa|password|publickey|keyboard-interactive/pam) for [^[:space:]]+ from [^[:space:]]+ port [0-9]+( (ssh|ssh2))?`

An AxoSyslog 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.

14.6.1.1 The structure of the pattern database

The pattern database is organized as follows:



- 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 `${PROGRAM}` macro) is compared to the Program Patterns of the available rulesets, and then the rules of the matching rulesets are applied to the message. (If the content of the `${PROGRAM}` macro is not the proper name of the application, you can use the `program-template()` option to specify it.)

- The Program Pattern can be a string that specifies the name of the application 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 AxoSyslog parsers used to segment messages. Additionally, every rule has a unique identifier: if a message matches a rule, the identifier of the rule is stored together with the message.

- Rules consist of a message pattern and a class. The Message Pattern is similar to the Program Pattern, but is applied to the message part of the log message (the content of the ${MESSAGE} macro). If a message pattern matches the message, the class of the rule is assigned to the message (for example, Security, Violation, and so on).

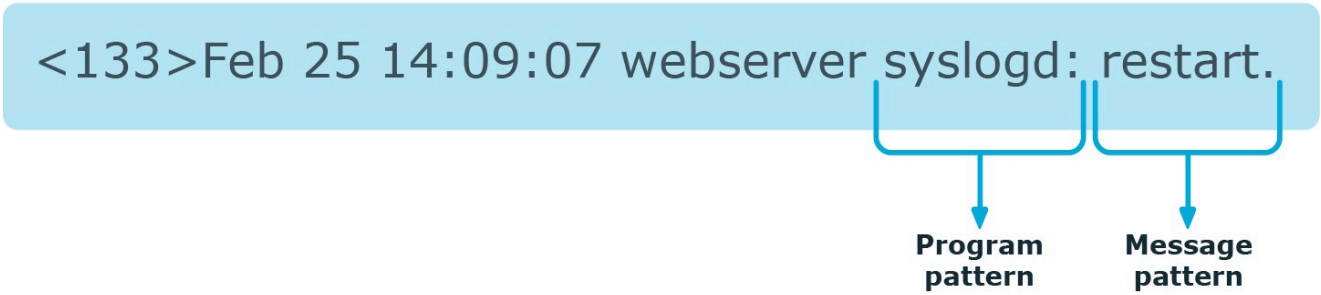
- Rules can also contain additional information about the matching messages, such as the description of the rule, an URL, name-value pairs, or free-form tags.

- Patterns can consist of literals (keywords, or rather, keycharacters) and pattern parsers.

> **Note**
>
> If the `${PROGRAM}` part of a message is empty, rules with an empty Program Pattern are used to classify the message.
>
> If the same Program Pattern is used in multiple rulesets, the rules of these rulesets are merged, and every rule is used to classify the message. Note that message patterns must be unique within the merged rulesets, but the currently only one ruleset is checked for uniqueness.
>
> If the content of the `${PROGRAM}` macro is not the proper name of the application, you can use the `program-template()` option to specify it.

14.6.1.2 How pattern matching works

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 are enclosed between `@` characters.**

When a new message arrives, AxoSyslog attempts to classify it using the pattern database. The available patterns are organized alphabetically into a tree, and AxoSyslog 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, AxoSyslog selects the patterns that start with `Ap`, and so on, until there is no more specific pattern left. The AxoSyslog 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`), AxoSyslog 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.

14.6.1.3 Artificial ignorance

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

The AxoSyslog 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 – AxoSyslog 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.

14.6.2 Using pattern databases

To classify messages using a pattern database, include a `db-parser()` statement in your `syslog-ng.conf` 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);
 };
```

By default, AxoSyslog tries to apply the patterns to the body of the incoming messages, that is, to the value of the $MESSAGE macro. If you want to apply patterns to a specific field, or to an expression created from the log message (for example, using template functions or other parsers), use the `message-template()` option. For example:

```
parser pattern_db {
     db-parser(
         file("/opt/syslog-ng/var/db/patterndb.xml")
         message-template("${MY-CUSTOM-FIELD-TO-PROCESS}")
     );
 };
```

By default, AxoSyslog uses the name of the application (content of the `${PROGRAM}` macro) to select which rules to apply to the message. If the content of the `${PROGRAM}` macro is not the proper name of the application, you can use the `program-template()` option to specify it. For example:

```
parser pattern_db {
     db-parser(
         file("/opt/syslog-ng/var/db/patterndb.xml")
         program-template("${MY-CUSTOM-FIELD-TO-SELECT-RULES}")
     );
 };
```

Note that the `program-template()` option is available in AxoSyslog version 3.21 and later.

> **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.**

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)
        );
    };
```

Note that if you chain pattern databases, that is, use multiple databases in the same log path, the class assigned to the message (the value of `${.classifier.class}`) will be the one assigned by the last pattern database. As a result, a message might be classified as unknown even if a previous parser successfully classified it. For example, consider the following configuration:

```
   log {
        ...
        parser(db_parser1);
        parser(db_parser2);
        ...
    };
```

Even if `db_parser1` matches the message, `db_parser2` might set `${.classifier.class}` to unknown. To avoid this problem, you can use an 'if' statement to apply the second parser only if the first parser could not classify the message:

```
   log {
        ...
        parser{ db-parser(file("db_parser1.xml")); };
        if (match("^unknown$" value(".classifier.class"))) {
            parser { db-parser(file("db_parser2.xml")); };
        };
        ...
    };
```

For details on how to create your own pattern databases see The 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 AxoSyslog version 3.11 and later.

14.6.2.1 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 AxoSyslog 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`.

14.6.2.2 Correlating log messages using pattern databases

The AxoSyslog application can correlate log messages identified using pattern databases . Alternatively, you can also correlate log messages using the `grouping-by()` parser. For details, see Correlating messages using the grouping-by() parser.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix email server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message. Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages with AxoSyslog, 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 AxoSyslog version 3.5, if a message is added to a context, AxoSyslog 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 AxoSyslog is reloaded (SIGHUP), but are lost when AxoSyslog is restarted.**

Another parameter of a rule is the `context-timeout` attribute, which determines how long a context is stored, that is, how long AxoSyslog waits for related messages to arrive.

Note the following points about timeout values:

- When a new message is added to a context, AxoSyslog will restart the timeout using the `context-timeout` set for the new message.
- When calculating if the timeout has already expired or not, AxoSyslog 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 AxoSyslog 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), AxoSyslog will replace its timestamp with the current system time.

Example: How AxoSyslog 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 AxoSyslog receives the messages within 1 second, the timeout event will occur immediately, because the difference of the two timestamp (60 seconds) is larger than the timeout value (10 seconds).

- 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.

14.6.2.2.1 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}` .

Warning

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 , the context-lookup , or the context-values template functions.

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 .

14.6.3 Options of db-parser() parsers

The `db-parser()` parser has the following options.

drop-unmatched()

Accepted values:	`yes` , `no`
Default:	`no`

Description: If set to `yes` , AxoSyslog discards every message that does not match any pattern in the database. For details and examples, see Using pattern databases.

file()

Synopsis:	`file(<path-to-pattern-database>)`
Default:	`/opt/syslog-ng/var/run/patterndb.xml`

Description: Path to the pattern database XML file to use. Different `db-parser()` statements can reference different pattern databases by setting different `file()` paths.

inject-mode()

Accepted values:	`internal` , `pass-through` , `aggregate-only`
Default:	`pass-through`

Description: Controls how AxoSyslog routes messages generated by the parser (for example, by `create-context` actions or trigger-action rules). The following values are available:

- *internal*: Inject generated messages into the internal source of AxoSyslog, so they appear in every log path that uses `source(s_internal)` .
- *pass-through*: Pass generated messages into the same log path as the original message that triggered them. This is the default.

- *aggregate-only*: Only emit the generated messages and drop the triggering original messages.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

message-template()

Synopsis:	`message-template("<template>")`
Default:	`${MESSAGE}`

Description: The template or macro that AxoSyslog evaluates and uses as input for pattern matching. By default, patterns are matched against the body of the message (`${MESSAGE}`). Use this option to apply patterns against a different field or against an expression built from the log message. The `template()` option is a generic alias for `message-template()` .

persist-name()

Synopsis:	`persist-name("<name>")`

Description: Sets a unique name for the parser instance, used to identify its state in the persist file across restarts. Set `persist-name()` if you have multiple `db-parser()` instances and want to ensure that state restoration works correctly after a configuration change.

prefix()

Synopsis:	`prefix("<prefix>")`

Description: Inserts a prefix before the name part of the name-value pairs that the parser sets, so that downstream processing can address them uniquely. For example, with `prefix(".patterndb.")` the parsed value `username` becomes `${.patterndb.username}`.

program-template()

Synopsis:	`program-template("<template>")`
Default:	`${PROGRAM}`

Description: The template that AxoSyslog evaluates to select which pattern database ruleset to apply. By default, the application name from the `${PROGRAM}` macro is used. Set this option if the program name is stored in a different macro or needs to be constructed from other fields.

Available in AxoSyslog version 3.21 and later.

template()

Synopsis:	`template("<template>")`
Default:	`${MESSAGE}`

Description: The generic parser-level alias for `message-template()`. The template or macro that AxoSyslog evaluates and uses as input for pattern matching. If both `template()` and `message-template()` are set, the last value wins.

14.6.4 Triggering actions for identified messages

The AxoSyslog 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 AxoSyslog 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.

14.6.4.1 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>
```

14.6.4.2 External actions

To perform an external action when a message is triggered, for example, to send the message in an email, you have to route the generated messages to an external application using the `program()` destination.

Example: Sending triggered messages to external applications

The following sample configuration selects the triggered messages and sends them to an external script.

1. Set a field in the triggered message that is easy to identify and filter. For example:

```
    <values>
        <value name="MESSAGE">A log message from ${HOST} matched rule number
$.classifier.rule_id</value>
        <value name="TRIGGER">yes</value>
    </values>
```

2. Create a destination that will process the triggered messages.

```
    destination d_triggers {
        program("/bin/myscript"; );
    };
```

3. Create a filter that selects the triggered messages from the internal source.

```
    filter f_triggers {
        match("yes" value ("TRIGGER") type(string));
    };
```

4. Create a logpath that selects the triggered messages from the internal source and sends them to the script:

```
log { source(s_local); filter(f_triggers); destination(d_triggers); };
```

5. Create a script that will actually process the generated messages, for example:

```
#!/usr/bin/perl
while (<>) {
    # body of the script to send emails, snmp traps, and so on
}
```

14.6.4.3 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 AxoSyslog 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`, AxoSyslog 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 AxoSyslog to send an email alert if a client stops sending messages.

- Configure your clients to send MARK messages periodically. It is enough to configure MARK messages for the destination that forwards your log messages to your AxoSyslog server (`mark-mode(periodical)`).
- On your AxoSyslog server, create a pattern database rule that matches on the incoming MARK messages. In the rule, set the `context-scope` attribute to `host` , and the `context-timeout` attribute to a value that is higher than the `mark-freq` value set on your clients (by default, `mark-freq` is 1200 seconds, so set `context-timeout` at least to 1500 seconds, but you might want to use a higher value, depending on your environment).
- Add an action to this rule that sends you an email alert if the `context-timeout` expires, and the server does not receive a new MARK message (`<action trigger="timeout">`).
- On your AxoSyslog server, use the pattern database in the log path that handles incoming log messages.

14.6.5 Creating pattern databases

14.6.5.1 Using pattern parsers

Pattern parsers attempt to parse a part of the message using rules specific to the type of the parser. Parsers are enclosed between @ characters. The syntax of parsers is the following:

- a beginning `@` character,
- the type of the parser written in capitals,
- optionally a name,
- parameters of the parser, if any, 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: @"`.

14.6.5.1.1 Pattern parsers

The following parsers are available in AxoSyslog. The internal parsers (for example, `@NUMBER@`) automatically associate type information to the parsed name-value pair. For details on data types, see Specifying data types in value-pairs.

@ANYSTRING@

Parses everything to the end of the message, you can use it to collect everything that is not parsed specifically to a single macro. In that sense its behavior is similar to the `greedy()` option of the CSV parser.

@DOUBLE@

An obsolete alias of the `@FLOAT@` parser.

@EMAIL@

This parser matches an email address. The parameter is a set of characters to strip from the beginning and the end of the email address. That way email addresses enclosed between other characters can be matched easily (for example, `<user@example.com>` or `"user@example.com"`. Characters that are valid for a hostname are not stripped from the end of the hostname. This includes a trailing period if present.

For example, the `@EMAIL:email:"[<]>` parser will match any of the following email addresses: `<user@example.com>`, `[user@example.com]`, `"user@example.com"`, and set the value of the `email` macro to `user@example.com`.

@ESTRING@

This parser has a required parameter that acts as the stopcharacter: the parser parses everything until it finds the stopcharacter. For example, to stop by the next `"` (double quote) character, use `@ESTRING::"@`. You can use the colon (`:`) as stopcharacter as well, for example: `@ESTRING:::@`. You can also specify a stopstring instead of a single character, for example, `@ESTRING::stop_here.@`. The `@` character cannot be a stopcharacter, nor can line-breaks or tabs.

@FLOAT@

A floating-point number that may contain a dot (.) character. (Up to AxoSyslog 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: @NLSTRING:.winaudit.SubjectUserSid@
        Account Name: @NLSTRING:.winaudit.SubjectUserName@
        Account Domain: @NLSTRING:.winaudit.SubjectDomainName@
        Logon ID: @NLSTRING:.winaudit.SubjectLogonId@

        Process Information:
        New Process ID: @NLSTRING:.winaudit.NewProcessId@
        New Process Name: @NLSTRING:.winaudit.NewProcessName@
        Token Elevation Type: @NLSTRING:.winaudit.TokenElevationType@
        Creator Process ID: @NLSTRING:.winaudit.ProcessId@
        Process Command Line: @NLSTRING:.winaudit.CommandLine@

        Token Elevation Type indicates the type of token that was assigned to the
new process in accordance with User Account Control policy.</pattern>
```

@NUMBER@

A sequence of decimal (0-9) numbers (for example, 1, 0687, and so on). Note that if the number starts with the 0x characters, it is parsed as a hexadecimal number, but only if at least one valid character follows 0x. A leading hyphen (`-`) is accepted for non-

hexadecimal numbers, but other separator characters (for example, dot or comma) are not. To parse floating-point numbers, use the @FLOAT@ parser.

@OPTIONALSET@

Parse any combination of the specified characters. 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 `@OPTIONALSET:: "@` parser will parse any combination of whitespaces and double-quotes.

Available in 3.31 and later.

> **Note**
>
> **The `@OPTIONALSET@` parser works almost exactly like the `@SET@` parser, but the `@OPTIONALSET@` parser continues parsing even if none of the specified characters are found.**

@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 AxoSyslog 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.

14.6.5.2 What's new in the 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.

14.6.5.3 The pattern database format

Pattern databases are XML files that contain rules describing the message patterns.

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 .

> **Note**
>
> Use the `pdbtool` utility that is bundled with AxoSyslog 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>
```

14.6.5.3.1 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'>
```

14.6.5.3.2 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'>
```

14.6.5.3.3 Element: patterns

Location

/patterndb /ruleset /*patterns*

Description

A container element. A `<patterns>` element may contain any number of `<pattern>` elements.

Attributes

N/A

Children

- *pattern*: The name of the application – syslog-ng matches this value to the ${PROGRAM} header of the syslog message to find the rulesets applicable to the syslog message.

  Specifying multiple patterns is useful if two or more applications have different names (that is, different ${PROGRAM} fields), but otherwise send identical log messages.

  It is not necessary to use multiple patterns if only the end of the ${PROGRAM} fields is different, use only the beginning of the ${PROGRAM} field as the `pattern`. For example, the Postfix email server sends messages using different process names, but all of them begin with the `postfix` string.

  You can also use parsers in the program pattern if needed, and use the parsed results later. For example: `<pattern>postfix\@ESTRING:.postfix.component:[@</pattern>`

  > **Note**
  >
  > **If the `<pattern>` element of a ruleset is not specified, AxoSyslog 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>
```

14.6.5.3.4 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>
```

14.6.5.3.5 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 Axoflow, or added to the XML by a local user.
- *id*: The globally unique ID of the rule.
- *class*: The class of the rule – this class is assigned to the messages matching a pattern of this rule.

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'>
```

14.6.5.3.6 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>
```

14.6.5.3.7 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

14.6.5.3.8 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 .

  You can associate types with values using the `"type"` attribute, for example, `integer` is a type-cast that associates `$foobar` with an integer type. For details on data types, see Specifying data types in value-pairs.

  ```
  <value name="foobar" type="integer">$PID</value>
  ```

  `db-parser()` 's internal parsers (for example, `@NUMBER@`) automatically associate type information to the parsed name-value pair.

  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>
```

14.6.5.3.9 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>
```

14.6.5.3.10 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*: An AxoSyslog 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, AxoSyslog 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 AxoSyslog 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` : AxoSyslog 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 AxoSyslog 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` , AxoSyslog collects every name-value pair from each message stored in the context, and includes them in the generated message. If a name-value pair appears in multiple messages of the context, the value in the latest message will be used. Note that tags are not merged, the generated message will inherit the tags assigned to the last message of the context.

      For details on the message context, see Correlating log messages using pattern databases and Actions and message correlation. For details on triggering messages, see Triggering actions for identified messages

      This option is available in AxoSyslog 5.3.2 and later.

- *values*: A container element for values and fields that are used to create the message generated by the action.
    - *value*: Sets the value of the message field specified in the `name` attribute of the element. For example, to specify the body of the generated message, use the following syntax:

      ```
          <value name="MESSAGE">A log message matched rule number
      $.classifier.rule_id</value>
      ```

      Note that currently it is not possible to add DATE, FACILITY, or SEVERITY fields to the message.

      When the action is used together with message correlation, the AxoSyslog 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: 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>
```

14.6.5.3.11 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 AxoSyslog version 3.8 and later.

Attributes

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` : AxoSyslog 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>
```

14.6.5.3.12 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 AxoSyslog 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>
```

14.6.5.3.13 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"
type="string">enabled</test_value>
```

- *name*: The name of the name-value pair to test.
- *type*: The type of the name-value pair, one of the recognized syslog-ng type hints Specifying data types in value-pairs

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"
type="integer">42156</test_value>
            </test_values>
        </example>
    </examples>
```

14.6.5.3.14 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>
```

14.7 Parsing enterprise-wide message model (EWMM) messages

The `ewmm-parser()` can be used to parse messages sent by another AxoSyslog 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.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

Declaration:

```
parser parser_name {
     ewmm-parser();
 };
```

14.8 Fortigate parser

The Fortigate parser can parse the log messages of FortiGate/FortiOS (Fortigate Next-Generation Firewall (NGFW)). These messages do not completely comply with the syslog RFCs, making them difficult to parse. The `fortigate-parser()` of AxoSyslog solves this problem, and can separate these log messages to name-value pairs. For details on using value-pairs, see Structuring macros, metadata, and other value-pairs. The parser can parse messages in the following format:

```
<PRI><NAME=VALUE PAIRS>
```

For example:

```
<189>date=2021-01-15 time=12:58:59 devname="FORTI_111" devid="FG100D3G12801312"
logid="0001000014" type="traffic" subtype="local" level="notice" vd="root"
eventtime=1610704739683510055 tz="+0300" srcip=91.234.154.139
srcname="91.234.154.139" srcport=45295 srcintf="wan1" srcintfrole="wan"
dstip=213.59.243.9 dstname="213.59.243.9" dstport=46730 dstintf="unknown0"
dstintfrole="undefined" sessionid=2364413215 proto=17 action="deny" policyid=0
policytype="local-in-policy" service="udp/46730" dstcountry="Russian Federation"
srccountry="Russian Federation" trandisp="noop" app="udp/46730" duration=0
sentbyte=0 rcvdbyte=0 sentpkt=0 appcat="unscanned" crscore=5 craction=262144
crlevel="low"
```

If you find a message that the `fortigate-parser()` cannot properly parse, contact us , so we can improve the parser.

By default, the Fortigate-specific fields are extracted into name-value pairs prefixed with `.fortigate.` For example, the devname in the previous message becomes `${.fortigate.devname}.` You can change the prefix using the prefix option of the parser.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `fortigate-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/fortigate/fortigate.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { network(transport("udp") flags(no-parse)); };
     parser { fortigate-parser(); };
     destination { ... };
 };
```

Note that you have to disable message parsing in the source using the `flags(no-parse)` option for the parser to work.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `websense-parser()` uses the `.websense.` prefix. To modify it, use the following format:

```
parser {
     websense-parser(prefix("myprefix."));
 };
```

14.8.1 Fortigate parser options

The `fortigate-parser()` has the following options:

prefix()

Synopsis:	prefix()
Default:	“.panos.”

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `fortigate-parser()` uses the `.fortigate.` prefix. To modify it, use the following format:

```
parser {
     fortigate-parser(prefix("myprefix."));
 };
```

template()

Synopsis:	`template("${<macroname>}")`

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

14.9 group-lines parser

Available in AxoSyslog version 4.2 and newer.

The `group-lines()` parser correlates multi-line messages received as separate, but subsequent lines into a single log message. AxoSyslog first collects the received messages into streams of related messages (based on the `key()` parameter), then grouped into correlation contexts up to `timeout()` seconds long. Multi-line messages are then identified within these contexts.

```
group-lines(
  key("$FILE_NAME")
  multi-line-mode("smart")
  template("$MESSAGE")
  timeout(10)
  line-separator("\n")
);
```

The parser has the following options.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

key()

Type:	template
Default:	

Description: Specifies a template that determines which messages form a single stream. Messages where the template expansion results in the same key are considered part of the same stream. Using the `key()` option, you can extract multi-line messages even if different streams are interleaved in your input.

line-separator()

Type:	string
Default:	`\n`

Description: In case a multi-line message is found, this string is inserted between the of the new multi-line message. Defaults to the newline character.

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, AxoSyslog 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()`, AxoSyslog 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`.

> **Warning**
>
> **If the `multi-line-garbage()` option is set, AxoSyslog discards lines between the line matching the `multi-line-garbage()` and the next line matching `multi-line-prefix()`.**

multi-line-mode()

Type:	`indented` , `prefix-garbage` , `prefix-suffix` , `regexp` , `smart`
Default:	empty string

Description: Use the `multi-line-mode()` option when processing multi-line messages. The AxoSyslog application provides the following methods to process multi-line messages:

- `indented` : 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.

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

- `prefix-garbage` : 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.

- `prefix-suffix` : 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.

- `smart` : The `smart` mode recognizes multi-line data backtraces even if they span multiple lines in the input. The backtraces are converted to a single log message for easier analysis. Backtraces for the following programming languages are recognized : Python, Java, JavaScript, PHP, Go, Ruby, and Dart.

`smart` mode is available in AxoSyslog version 4.2 and newer.

The regular expressions to recognize these programming languages are specified in an external file called `/usr/share/syslog-ng/smart-multi-line.fsm` (installation path depends on configure arguments), in a format that is described in that file.

Note

To format multi-line messages to your individual needs, consider the following:

- 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, AxoSyslog ignores newline characters

from the source until a line matches the regular expression again, and treats the lines between the matching lines as a single message. See also the `multi-line-garbage()` option.

Note

To format multi-line messages to your individual needs, consider the following:

- 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:2
5)
           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:2
5)
           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 AxoSyslog 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.

scope()

Type:	`process`, `program`, `host`, or `global`
Default:	`global`

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.

sort-key()

Type:	template
Default:	

Description: Sorts the lines of the correlation context before AxoSyslog assembles them into a single multi-line message. Use this option when the lines can arrive out of order. Note that specifying several macros or a complex template in `sort-key()` can slow down AxoSyslog.

template()

Type:	template
Default:	

Description: A template string that specifies what constitutes an line to `group-lines()` . In simple cases this is `${MSG}` or `${RAWMSG}` .

timeout()

Type:	number (seconds)
Default:	

Description: Specifies the maximum time to wait for the remaining lines of a multi-line message. If no new line is added to the correlation context during this period, AxoSyslog considers the multi-line message complete and forwards it. If a new line is added to the context, the timeout period restarts.

14.10 iptables parser

The iptables parser can parse the log messages of the iptables command.

FilterX has no iptables parser, but you can reimplement this one in a FilterX block. For details, see Create an iptables parser .

Prerequisites

- AxoSyslog version 3.16 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `iptables-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/iptables/iptables.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { system(); };
     parser { iptables-parser(); };
     destination { ... };
 };
```

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `iptables-parser()` uses the `.iptables.` prefix. To modify it, use the following format:

```
parser {
     iptables-parser(prefix("myprefix."));
 };
```

14.11 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 AxoSyslog application can separate parts of incoming JSON-encoded log messages to name-value pairs. For details on using value-pairs in AxoSyslog 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, AxoSyslog converts it to dot-notation-format. For example, to access the value of the following structure `{"KEY1": {"KEY2": "VALUE"}}`, use the `${KEY1.KEY2}` macro.

In FilterX, use the related `json()` cast to parse a JSON string into a dict. Unlike `json-parser()`, it does not create name-value pairs, and has no `prefix()` or `extract-prefix()` option.

Warning

If the names of keys in the JSON content are the same as the names of AxoSyslog soft macros, the value from the JSON content will overwrite the value of the macro. For example, the `{"PROGRAM":"value1","MESSAGE":"value2"}` JSON content will overwrite the `${PROGRAM}` and `${MESSAGE}` macros. To avoid overwriting such macros, use the `prefix()` option.

Hard macros cannot be modified, so they will not be overwritten. For details on the macro types, see Hard versus soft macros.

Note

When using the `json-parser()`, AxoSyslog converts all elements of the JSON object to name-value pairs. Any type information originally present in the incoming JSON object is retained, and automatically propagated to other AxoSyslog components (for example, a destination) if they support types.

- Elements without a type are treated as strings.
- JSON lists (arrays) are converted to AxoSyslog lists, so you can manipulate them using the `$(list-*)` template functions .

Note that prior to version 4.0, AxoSyslog handled every data as strings.

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 AxoSyslog 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);
 };
```

14.11.1 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"));
         };
     };
 };
```

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

key-delimiter()

Type:	character
Default:	`.`

Description: The delimiter character to use when parsing flattened keys. Supports Only single characters.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

This parser does not have a default prefix. To configure a custom prefix, use the following format:

```
parser {
     json-parser(prefix("myprefix."));
 };
```

template()

Synopsis: `template("${<macroname>}")`

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

14.12 Parsing key=value pairs

The AxoSyslog 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 AxoSyslog 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 AxoSyslog 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}` .

See also the equivalent FilterX function, `parse_kv()` .

Note

If a log message contains the same key multiple times (for example, `key1=value1, key2=value2, key1=value3, key3=value4, key1=value5`), then AxoSyslog only stores the last (rightmost) value for the key. Using the previous example, AxoSyslog will store the following pairs: `key1=value5, key2=value2, key3=value4` .

Warning

If the names of keys in the message is the same as the names of AxoSyslog soft macros, the value from the parsed message will overwrite the value of the macro. For example, the `PROGRAM=value1, MESSAGE=value2` content will overwrite the `${PROGRAM}` and `${MESSAGE}` macros. To avoid overwriting such macros, use the `prefix()` option.

Hard macros cannot be modified, so they will not be overwritten. For details on the macro types, see Hard versus soft macros.

The parser discards message sections that are not `key=value` pairs, even if they appear between `key=value` pairs that can be parsed.

The names of the keys can contain only the following characters: numbers (0-9), letters (a-z,A-Z), underscore (_), dot (.), hyphen (-). Other special characters are not permitted.

To parse `key=value` pairs, define a parser that has the `kv-parser()` option. Defining the prefix is optional. By default, the parser will process the `${MESSAGE}` part of the log message. You can also define the parser inline in the log path.

Declaration:

```
parser parser_name {
     kv-parser(
         prefix()
     );
 };
```

Example: Using a key=value parser

In the following example, the source is a log message consisting of comma-separated `key=value` pairs, for example, a Postfix log message:

```
Jun 20 12:05:12 mail.example.com <info> postfix/qmgr[35789]: EC2AC1947DA: from=
<me@example.com>, size=807, nrcpt=1 (queue active)
```

The kv-parser inserts the “ `.kv.` ” prefix before all extracted name-value pairs. The destination is a file, that uses the `format-json` template function. Every name-value pair that begins with a dot (" `.` ") character will be written to the file (`dot-nv-pairs`). The log line connects the source, the destination and the parser.

```
source s_kv {
     network(port(21514));
 };

 destination d_json {
     file("/tmp/test.json"
         template("$(format-json --scope dot-nv-pairs)\n"));
 };

 parser p_kv {
     kv-parser (prefix(".kv."));
 };

 log {
     source(s_kv);
     parser(p_kv);
     destination(d_json);
 };
```

You can also define the parser inline in the log path.

```
source s_kv {
     network(port(21514));
 };

 destination d_json {
     file("/tmp/test.json"
         template("$(format-json --scope dot-nv-pairs)\n"));
 };

 log {
     source(s_kv);
     parser {
         kv-parser (prefix(".kv."));
     };
     destination(d_json);
 };
```

You can set the separator character between the key and the value to parse for example, `key:value` pairs, like MySQL logs:

```
   Mar  7 12:39:25 myhost MysqlClient[20824]: SYSTEM_USER:'oscar',
MYSQL_USER:'my_oscar', CONNECTION_ID:23, DB_SERVER:'127.0.0.1', DB:'--', QUERY:'USE
test;'
```

```
   parser p_mysql {
        kv-parser(value-separator(":") prefix(".mysql."));
    };
```

14.12.1 Options of key=value parsers

The `kv-parser` has the following options.

allow-pair-separator-in-value()

Accepted values:	`yes` , `no`
Default:	`yes`

Description: Permits the pair separator character inside the value of a key-value pair, so that `key1=value with spaces, key2=other value` parses into two pairs even though the value of `key1` contains spaces.

AxoSyslog always parses key-value pairs this way, so setting this option has no effect. AxoSyslog only accepts it so that older configurations remain valid.

extract-stray-words-into()

Synopsis:	`extract-stray-words-into("<name-value-pair>")`

Description: Specifies the name-value pair where AxoSyslog 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 AxoSyslog 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`

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `kv-parser()` uses the `.kv.` prefix. To modify it, use the following format:

```
parser {
     kv-parser(prefix("myprefix."));
 };
```

pair-separator()

Synopsis: `pair-separator("<separator-string>")`

Description: Specifies the character or string that separates the key-value pairs from each other. Default value: `,` .

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(":"));` .

14.13 Linux audit parser

The Linux audit parser can parse the log messages of the Linux Audit subsystem (`auditd`). The AxoSyslog application can separate these log messages to name-value pairs. For details on using value-pairs in AxoSyslog 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 AxoSyslog application automatically decodes these fields (for example, the `c000003e` value becomes `x86_64`).

The AxoSyslog 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 AxoSyslog 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);
 };
```

14.13.1 Options of linux-audit-parser() parsers

The `linux-audit-parser()` has the following options. Because `linux-audit-parser()` is a specialized key-value parser, it accepts the same key-value options as Options of key=value parsers.

allow-pair-separator-in-value()

Accepted values:	`yes` , `no`
Default:	`yes`

Description: Permits the pair separator character inside the value of a key-value pair, so that `key1=value with spaces, key2=other value` parses into two pairs even though the value of `key1` contains spaces.

AxoSyslog always parses key-value pairs this way, so setting this option has no effect. AxoSyslog only accepts it so that older configurations remain valid.

extract-stray-words-into()

Synopsis:	`extract-stray-words-into("<name-value-pair>")`

Description: Specifies the name-value pair where AxoSyslog 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 AxoSyslog 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`

internal()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

pair-separator()

Synopsis:	`pair-separator("<separator-string>")`

Description: Specifies the character or string that separates the key-value pairs from each other. Default value: `,`.

For example, to parse `key1=value1;key2=value2` pairs, use `kv-parser(pair-separator(";"));`.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `linux-audit-parser()` uses the `.auditd.` prefix. To modify it, use the following format:

```
parser {
     linux-audit-parser(prefix("myprefix."));
 };
```

template()

Synopsis: `template("${<macroname>}")`

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

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(":"));` .

14.14 MariaDB parser

The MariaDB parser can parse the log messages of the MariaDB Audit Plugin. The parser supports the `syslog` output typess' format.

Prerequisites

- AxoSyslog version 3.37 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `mariadb-audit-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/mariadb/audit.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { system(); };
     parser { mariadb-audit-parser(); };
     destination { ... };
 };
```

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `mariadb-audit` uses the `.mariadb.` prefix. To modify it, use the following format:

```
parser {
     mariadb-audit-parser(prefix("myprefix."));
 };
```

template()

Synopsis: `template("${<macroname>}")`

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

14.15 metrics-probe

Available in AxoSyslog version 4.1.1 and newer.

`metrics-probe()` is a special parser that counts the messages that pass through the log path, and creates labeled stats counters based on the fields of the passing messages.

You can configure the name of the keys and the labels. Note that the keys are automatically prefixed with the `syslogng_` string. You can use templates in the values of the labels.

See also the equivalent FilterX function, `update_metric()` , which takes the counter name as its first argument instead of in a `key()` option.

The minimal configuration creates counters with the key `syslogng_classified_events_total` and labels `app` , `host` , `program` and `source` . For example:

```
parser p_metrics_probe {
  metrics-probe();

  # Same as:
  #
  # metrics-probe(
  #   key("classified_events_total")
  #   labels(
  #     "app" => "${APP}"
  #     "host" => "${HOST}"
  #     "program" => "${PROGRAM}"
  #     "source" => "${SOURCE}"
  #   )
  # );
};
```

This configuration results in counters like:

```
syslogng_classified_events_total{app="example-app", host="localhost",
program="baz", source="s_local_1"} 3
syslogng_classified_events_total{app="example-app", host="localhost",
program="bar", source="s_local_1"} 1
syslogng_classified_events_total{app="example-app", host="localhost",
program="foo", source="s_local_1"} 1
```

You can query the metrics by running the following command:

```
syslog-ng-ctl stats prometheus
```

For example, the following `metrics-probe()` parser creates a counter called `syslogng_custom_key` that counts messages that have their `custom_label_name_1` field set to `foobar`, and for these messages it creates separate counters based on the value of the `custom_label_name_2` field.

```
parser p_metrics_probe {
  metrics-probe(
    key("custom_key")  # adds "syslogng_" prefix => "syslogng_custom_key"
    labels(
      "custom_label_name_1" => "foobar"
      "custom_label_name_2" => "${.custom.field}"
    )
  );
};
```

This configuration results in counters like:

```
syslogng_custom_key{custom_label_name_1="foobar", custom_label_name_2="bar"} 1
syslogng_custom_key{custom_label_name_1="foobar", custom_label_name_2="foo"} 1
syslogng_custom_key{custom_label_name_1="foobar", custom_label_name_2="baz"} 3
```

Starting with AxoSyslog 4.4, you can create dynamic labels as well.

Options

The `metrics-probe()` parser has the following options. In addition to the options listed here, `metrics-probe()` accepts the `value-pairs()` options – such as `cast()`, `exclude()`, `include-bytes()`, `key()`, `pair()`, `rekey()`, and `scope()` – inside its `labels()` option. For details, see Dynamic labels.

frac-digits()

Type:	number
Default:	0

Description: The AxoSyslog application can store fractions of a second in the timestamps according to the ISO8601 format. The `frac-digits()` parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received.

> **Note**
>
> **The AxoSyslog application can add the fractions to non-ISO8601 timestamps as well.**

> **Note**
>
> **As AxoSyslog is precise up to the microsecond, when the `frac-digits()` option is set to a value higher than 6, AxoSyslog will truncate the fraction seconds in the timestamps after 6 digits.**

increment()

Type:	integer or template
Default:	1

Available in AxoSyslog version 4.2 and newer.

Sets a template, which resolves to a number that defines the increment of the counter. The following example defines a counter called `syslogng_input_event_bytes_total`, and increases its value with the size of the incoming message (in bytes).

```
metrics-probe(
    key("input_event_bytes_total")
    labels(
        "cluster" => "`cluster-name`"
        "driver" => "kubernetes"
        "id" => "${SOURCE}"
        "namespace" => "${`prefix`namespace_name}"
        "pod" => "${`prefix`pod_name}"
    )
    increment("${RAWMSG_SIZE}")
);
```

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

key()

Type:	string
Default:	`classified_events_total`

The name of the counter to create. Note that the value of this option is always prefixed with `syslogng_` , so for example `key("my-custom-key")` becomes `syslogng_my-custom-key` .

labels()

Type:	

Default: See the description

The labels used to create separate counters, based on the fields of the messages processed by `metrics-probe()`. Use the following format:

```
labels(
    "name-of-the-label-in-the-output" => "field-of-the-message"
)
```

Default value:

```
labels(
    "app" => "${APP}"
    "host" => "${HOST}"
    "program" => "${PROGRAM}"
    "source" => "${SOURCE}"
)
```

This results in counters like:

```
syslogng_classified_events_total{app="example-app", host="localhost",
program="baz", source="s_local_1"} 3
```

Dynamic labels

Available in AxoSyslog 4.4 and newer.

Dynamic labelling allows you to use every available `value-pairs()` options in the labels, for example, `key()`, `rekey()`, `pair()`, or `scope()`.

For example:

```
metrics-probe(
  key("foo")
  labels(
    "static-label" => "bar"
    key(".my_prefix.*" rekey(shift-levels(1)))
  )
);
```

```
syslogng_foo{static_label="bar",my_prefix_baz="anotherlabel",my_prefix_foo="bar",my_prefix_nested_axo="flow"} 4
```

level()

Type:	integer (0-3)
Default:	0

Available in AxoSyslog version 4.2 and newer.

Sets the stats level of the generated metrics.

> Note: Drivers configured with `internal(yes)` register their metrics on level 3. That way if you are creating an SCL, you can disable the built-in metrics of the driver, and create metrics manually using `metrics-probe()`.

local-time-zone()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Sets the timezone that AxoSyslog uses when it expands a timestamp macro in the `increment()` or `labels()` templates as a local time.

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()

Type:	**One of:** `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 AxoSyslog cannot convert some data to the specified type. By default, AxoSyslog 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 AxoSyslog.
- `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()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Sets the timezone that AxoSyslog uses when it expands a timestamp macro in the `increment()` or `labels()` templates. The `time-zone()` option is an alias of `send-time-zone()` .

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.

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}`).

template-escape()

Type:	`yes` , `no`
Default:	`no`

Description: Turns on escaping for the `'` , `"` , and backspace characters in the expanded value of the `increment()` and `labels()` templates.

time-zone()

Type:	name of the timezone, or the timezone offset
Default:	The local timezone.

Description: Alias of `send-time-zone()` .

ts-format()

Type:	`rfc3164` , `bsd` , `rfc3339` , `iso`
Default:	`rfc3164`

Description: Overrides the global `ts-format()` option for the timestamp macros expanded in the `increment()` and `labels()` templates.

14.16 Netskope parser

The Netskope parser can parse Netskope log messages. These messages do not completely comply with the syslog RFCs, making them difficult to parse. The `netskope-parser()` of AxoSyslog solves this problem, and can separate these log messages to name-value pairs. For details on using value-pairs in AxoSyslog see Structuring macros, metadata, and other value-pairs. The parser can parse messages in the following format:

```
<PRI>{JSON-formatted-log-message}
```

For example:

```
<134>{"count": 1, "supporting_data": {"data_values": ["x.x.x.x",
"user@domain.com"], "data_type": "user"}, "organization_unit":
"domain/domain/Domain Users/Enterprise Users", "severity_level": 2, "category":
null, "timestamp": 1547421943, "_insertion_epoch_timestamp": 1547421943, "ccl":
"unknown", "user": "user@domain.com", "audit_log_event": "Login Successful",
"ur_normalized": "user@domain.com", "_id": "936289", "type": "admin_audit_logs",
"appcategory": null}
```

If you find a message that the `netskope-parser()` cannot properly parse, contact us , so we can improve the parser.

The AxoSyslog application sets the `${PROGRAM}` field to `Netskope` .

By default, the Netskope-specific fields are extracted into name-value pairs prefixed with `.netskope` . For example, the `organization_unit` in the previous message becomes `${.netskope.organization_unit}` . You can change the prefix using the `prefix` option of the parser.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `netskope-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/netskope/plugin.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { network(flags(no-parse)); };
     parser { netskope-parser(); };
     destination { ... };
 };
```

Note that you have to disable message parsing in the source using the `flags(no-parse)` option for the parser to work.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `netskope-parser()` uses the `.netskope.` prefix. To modify it, use the following format:

```
parser {
     netskope-parser(prefix("myprefix."));
 };
```

14.17 Parse OpenTelemetry messages

By default, AxoSyslog doesn't parse the fields of incoming OpenTelemetry messages into name-value pairs, but are only available for forwarding using the `opentelemetry()` destination. To parse the fields into name-value pairs, use the `opentelemetry()` parser.

The `opentelemetry()` parser parses the fields into name-value pairs starting with the `.otel.` prefix.

- The type of the message is stored in the `.otel.type` field (possible values: `log`, `metric`, and `span`).
- Resource information is mapped into the `.otel.resource.<...>`, for example, `.otel.resource.dropped_attributes_count`, or `.otel.resource.schema_url`.
- Scope information is mapped into `.otel.scope.<...>`, for example, `.otel.scope.name`, `.otel.scope.schema_url`.
- The fields of log records are mapped into `.otel.log.<...>`, for example, `.otel.log.body`, `.otel.log.severity_text`.
- The fields of metrics are mapped into `.otel.metric.<...>`, for example, `.otel.metric.name`, `.otel.metric.unit`.
  - The type of the metric is mapped into `.otel.metric.data.type`. Possible values: `gauge`, `sum`, `histogram`, `exponential_histogram`, `summary`.
  - The actual data is mapped into `.otel.metric.data.<type>.<...>`, for example, `.otel.metric.data.gauge.data_points.0.time_unix_nano`.
- The fields of traces are mapped into `.otel.span.<...>`, for example, `.otel.span.name`, `.otel.span.trace_state`. Repeated fields have an index, for example, `.otel.span.events.5.time_unix_nano`.

For details on the parsed fields, you can check the OpenTelemetry proto files.

See also the related Handle OpenTelemetry log records functions. They map the message into typed OTEL objects instead of flattening it into `.otel.` name-value pairs.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-grpc` package on Debian/Ubuntu, or the `axosyslog-grpc` package on RHEL and compatible distributions. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error.

Mapping data types

String, bool, int64, double, and bytes values are mapped to their respective AxoSyslog name-value type, for example, `.otel.resource.attributes.string_key` becomes a string value.

The mapping of AnyValue type fields is limited.

`ArrayValue` and `KeyValueList` types are stored serialized with protobuf type. Note that `protobuf` and `bytes` types are only available, unless explicitly type cast. For example, `bytes(${.otel.log.span_id})`. When using template functions, use `--include-bytes`, for example, `$(format-json .otel.* --include-bytes`. In the case of `$(format-json)`, the content is base64-encoded into the bytes content.

Options

The `opentelemetry()` parser has the following options.

internal()

Accepted values:	`yes`, `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

set-hostname()

Synopsis:	`yes` or `no`
Default:	`yes`

Available in AxoSyslog 4.8 and later.

Description: If set to yes, the parser extracts the `host.name` resource attribute if available in the message. Otherwise, it leaves the HOST field as-is.

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}`).

14.18 panos-parser(): parsing PAN-OS log messages

The PAN-OS (a short version of Palo Alto Networks Operating System) parser can parse log messages originating from Palo Alto Networks devices. Even though these messages completely comply to the RFC standards, their `MESSAGE` part is not a plain text. Instead, the `MESSAGE` part contains a data structure that requires additional parsing.

The `panos-parser()` of AxoSyslog solves this problem, and can separate PAN-OS log messages to name-value pairs. For details on using value-pairs in AxoSyslog, see Structuring macros, metadata, and other value-pairs.

Prerequisites

- Version 3.29 of AxoSyslog or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `panos-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/paloalto/panos.conf on GitHub.
- PAN-OS log messages from Palo Alto Networks devices.

Limitations

The `panos-parser()` only works on AxoSyslog version 3.29 or later.

Configuration

You can include the `panos-parser()` in your AxoSyslog configuration like this:

```
parser p_parser{
     panos-parser();
 };
```

14.18.1 Message format parsed by panos-parser()

This section illustrates the most commonly used PAN-OS log format on the AxoSyslog side.

For information about customizing log format on the PAN-OS side, see the relevant section of the PAN-OS® Administrator's Guide .

Message format and log format

Using the `panos-parser()` , the parsed messages in AxoSyslog have the following general format:

```
<PRI><TIMESTAMP> <HOST> <PALO-ALTO-fields-in-CSV-format>
```

There are several "types" of log formats in Palo Alto Networks PAN-OS. For example, the most commonly used SYSTEM type has the following message format on the AxoSyslog side after parsing:

```
<12>Apr 14 16:48:54 paloalto.test.net 1,2020/04/14
16:48:54,unknown,SYSTEM,auth,0,2020/04/14 16:48:54,,auth-
fail,,0,0,general,medium,failed authentication for user 'admin'. Reason: Invalid
username/password. From: 10.0.10.55.,1718,0x0,0,0,0,0,,paloalto
```

14.18.2 PAN-OS parser options

The `panos-parser()` has the following options:

prefix()

Synopsis:	prefix()
Default:	".panos."

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

14.19 PostgreSQL csvlog

This parser processes messages in the PostgreSQL csvlog format. The following sample message is a multi-line message with embedded NL characters. This is a single, multi-line log entry that starts with the timestamp.

```
2023-08-08 12:05:52.805 UTC,,,22113,,64d22fa0.5661,1,,2023-08-08 12:05:52
UTC,23/74060,0,LOG,00000,"automatic vacuum of table ""tablename"": index scans: 0
pages: 0 removed, 4 remain, 0 skipped due to pins, 0 skipped frozen
tuples: 114 removed, 268 remain, 0 are dead but not yet removable, oldest xmin:
149738000
buffer usage: 97 hits, 0 misses, 6 dirtied
avg read rate: 0.000 MB/s, avg write rate: 114.609 MB/s
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s",,,,,,,,,""
```

Prerequisites

- AxoSyslog version 4.5.0 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `postgresql-csvlog-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/pgsql/pgsql.conf on GitHub.

Configuration

The `postgresql-csvlog-parser()` extracts the information from this message into a set of name-value pairs. By default, the name-value pairs have the `.pgsql` prefix.

```
@version: current

log {
    source { file("/var/log/pgsql.log" follow-freq(1) flags(no-parse)); };
    parser { postgresql-csvlog-parser() };
    destination { ... };
};
```

FilterX has no PostgreSQL parser. Since this driver builds on the CSV parser, the related `parse_csv()` function is the closest building block.

Options

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

on-type-error()

Synopsis:	string

Description: Specifies what to do when casting a parsed value to a specific data type fails. Note that the `flags(drop-invalid)` option and the `on-error()` global option also affects the behavior.

Accepts the same values as the `on-error()` global option .

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

14.20 Python parser

The Python log parser (available in AxoSyslog 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.

The following points apply to using Python blocks in AxoSyslog in general:

- Python parsers and template functions are available in AxoSyslog version 3.10 and later.

  Python destinations and sources are available in AxoSyslog version 3.18 and later.

- Supported Python versions: 2.7 and 3.4+ (if you are using pre-built binaries, check the dependencies of the package to find out which Python version it was compiled with).

- The Python block must be a top-level block in the AxoSyslog configuration file.

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

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

- The Python object is initiated every time when AxoSyslog is started or reloaded.

  > **Warning**
  >
  > **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

- The Python block can contain multiple Python functions.

- Using Python code in AxoSyslog can significantly decrease the performance of AxoSyslog, especially if the Python code is slow. In general, the features of AxoSyslog are implemented in C, and are faster than implementations of the same or similar features in Python.
- Validate and lint the Python code before using it. The AxoSyslog application does not do any of this.
- Python error messages are available in the `internal()` source of AxoSyslog.
- You can access the name-value pairs of AxoSyslog directly through a message object or a dictionary.
- To help debugging and troubleshooting your Python code, you can send log messages to the `internal()` source of AxoSyslog. For details, see Logging from your Python code.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-python` package on Debian/Ubuntu , or the `axosyslog-python` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Declaration:

Python parsers consist of two parts. The first is a AxoSyslog parser object that you use in your AxoSyslog configuration, for example, in the log path. This parser references a Python class, which is the second part of the Python parsers. The Python class processes the log messages it receives, and can do virtually anything that you can code in Python.

```
   parser <name_of_the_python_parser>{
        python(
            class("<name_of_the_python_class_executed_by_the_parser>")
        );
    };

    python {
    class MyParser(object):
        def init(self, options):
            '''Optional. This method is executed when syslog-ng is started or
reloaded.'''
            return True
        def deinit(self):
            '''Optional. This method is executed when syslog-ng is stopped or
reloaded.'''
            pass
        def parse(self, msg):
            '''Required. This method receives and processes the log message.'''
            return True
    };
```

Options of the python() parser

The `python()` parser has the following options.

class()

Type:	string
Default:	N/A

Description: The name of the Python class that implements the parser, for example:

```
   python(
        class("MyPythonParser")
    );
```

If you want to store the Python code in an external Python file, the `class()` option must include the name of the Python file containing the class, without the path and the `.py` extension, for example:

```
python(
     class("MyPythonfilename.MyPythonParser")
 );
```

For details, see Python code in external files

imports() (DEPRECATED)

Obsolete alias for `loaders()` . When set, behaves identically to `loaders()` . Use `loaders()` instead.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

loaders()

Type:	list of python modules
Default:	empty list

Description: The AxoSyslog application imports Python modules specified in this option, before importing the code of the Python class. This option has effect only when the Python class is provided in an external Python file. This option has no effect when the

Python class is provided within the AxoSyslog configuration file (in a `python{}` block). You can use the `loaders()` option to modify the import mechanism that imports Python class. For example, that way you can use hy in your Python class.

```
python(class(usermodule.HyParser) loaders(hy))
```

options()

Type:	string
Default:	N/A

Description: This option allows you to pass custom values from the configuration file to the Python code. Enclose both the option names and their values in double-quotes. The Python code will receive these values during initialization as the `options` dictionary. For example, you can use this to set the IP address of the server from the configuration file, so it is not hard-coded in the Python object.

```
python(
     class("MyPythonClass")
     options(
         "host" "127.0.0.1"
         "port" "1883"
         "otheroption" "value")
 );
```

For example, you can refer to the value of the host field in the Python code as `options["host"]` . Note that the Python code receives the values as strings, so you might have to cast them to the type required, for example: `int(options["port"])`

> **Note**
>
> From version 3.27, AxoSyslog supports the arrow syntax for declaring custom Java and Python options. You can alternatively declare them using a similar syntax:

```
options(
  "host" => "localhost"
  "port" => "1883"
  "otheroption" => "value"
)
```

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}`).

Methods of the python() parser

The init (self, options) method (optional)

The AxoSyslog application initializes Python objects only when it is started or reloaded. That means it keeps the state of internal variables while AxoSyslog 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, AxoSyslog 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 AxoSyslog will not start.

The return value of the `parse()` method must be `True`. If it returns `False`, or raises an exception, AxoSyslog 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 AxoSyslog configuration file (for example, in a template), use the `${new-macro-name}` macro.

- You cannot override hard macros (see Hard versus soft macros).

- To list all available keys (names of name-value pairs), use the `log_message.keys()` function.

The deinit(self) method (optional)

This method is executed when AxoSyslog is stopped or reloaded.

> **Warning**
>
> **If you reload AxoSyslog, existing Python objects are destroyed, therefore the context and state information of Python blocks is lost. Log rotation and updating the configuration of AxoSyslog typically involves a reload.**

Example: Parse loggen logs

The following sample code parses the messages of the `loggen` tool (for details, see The loggen manual page). The following is a sample loggen message:

```
   <38>2017-04-05T12:16:46 localhost prg00000[1234]: seq: 0000000000, thread: 0000,
runid: 1491387406, stamp: 2017-04-05T12:16:46
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPAD
DPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD
```

The AxoSyslog 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 AxoSyslog log statement uses the extracted fields to format the output message.

```
   @version: 4.29
    @include "scl.conf"
    parser my_python_parser{
        python(
            class("LoggenParser")
            options("regex", "seq: (?P<seq>\\d+), thread: (?P<thread>\\d+), runid:
(?P<runid>\\d+), stamp: (?P<stamp>[^ ]+) (?P<padding>.*$)")
        );
    };
    log {
        source { tcp(port(5555)); };
        parser(my_python_parser);
        destination {
            file("/tmp/regexparser.log.txt" template("seq: $seq thread: $thread
runid: $runid stamp: $stamp my_counter: $MY_COUNTER"));
        };
    };
    python {
    import re
    class LoggenParser(object):
        def init(self, options):
            pattern = options["regex"]
            self.regex = re.compile(pattern)
            self.counter = 0
            return True
        def deinit(self):
            pass
        def parse(self, log_message):
            match = self.regex.match(log_message['MESSAGE'])
            if match:
                for key, value in match.groupdict().items():
                    log_message[key] = value
                log_message['MY_COUNTER'] = self.counter
                self.counter += 1
                return True
            return False
    };
```

Example: Parse Windows eventlogs in Python - performance

The following example uses regular expressions to process Windows log messages received in XML format. The parser extracts different fields from messages received from the Security and the Application eventlog containers. Using the following configuration file, AxoSyslog could process about 25000 real-life Windows log messages per second.

```
   @version: 4.29
    options {
        keep-hostname(yes);
        keep-timestamp(no);
        stats-level(2);
        use-dns(no);
    };
    source s_network_aa5fdf25c39d4017a8e504cdb641b477 {
        network(
            flags(no-parse)
            ip(0.0.0.0)
            log-fetch-limit(1000)
            log-iw-size(100000)
            max-connections(100)
            port(514)
        );
    };
    parser p_python_parser_79c31da44bb64de6b5de84be4ae15a15 {
        python(options("regex_for_security", ".* Security ID:  (?
P<security_id>\\S+)   Account Name:  (?P<account_name>\\S+)   Account Domain:  (?
P<account_domain>\\S+)   Logon ID:  (?P<logon_id>\\S+).*Process Name: (?
P<process_name>\\S+).*EventID (?P<event_id>\\d+)", "regex_others", "(.*)EventID (?
P<event_id>\\d+)")
    class("EventlogParser"));
    };
    destination d_file_78363e1dd90c4ebcbb0ee1eff5a2e310 {
        file(
            "/var/testdb_working_dir/fcd713a2-d48e-4025-9192-ec4a9852cafa.$HOST"
            flush-lines(1000)
            log-fifo-size(200000)
        );
    };
```

```
log {
    source(s_network_aa5fdf25c39d4017a8e504cdb641b477);
    parser(p_python_parser_79c31da44bb64de6b5de84be4ae15a15);
    destination(d_file_78363e1dd90c4ebcbb0ee1eff5a2e310);
    flags(flow-control);
};

python {
import re
class EventlogParser(object):
    def init(self, options):
        self.regex_security = re.compile(options["regex_for_security"])
        self.regex_others = re.compile(options["regex_others"])
        return True
    def deinit(self):
        pass
    def parse(self, log_message):
        security_match = self.regex_security.match(log_message['MESSAGE'])
        if security_match:
            for key, value in security_match.groupdict().items():
                log_message[key] = value
        else:
            others_match = self.regex_others.match(log_message['MESSAGE'])
            if others_match:
                for key, value in others_match.groupdict().items():
                    log_message[key] = value
        return True
};
```

14.21 Regular expression (regexp) parser

The AxoSyslog application can parse fields from a message with the help of regular expressions. This can be also achieved with the `match()` filter, by setting the store-matches flag, but the `regexp-parser()` offers more flexibility, like multiple patterns and setting the prefix of the created name-value pairs.

> **Note**
>
> **The `regexp-parser()` can create additional name-value pairs only if "named capture groups" are used in the regular expression, for example `(?<test_field>\w+)`. For more information, see "named capture groups" in PCRE documentation .**

For more information about regular expressions in AxoSyslog, see Regular expressions.

See also the related FilterX function `regexp_search()` , which returns the matches instead of creating name-value pairs, and takes a single pattern instead of a list.

For example:

Declaration:

```
parser p_regexp {
     regexp-parser(
     patterns( ... )
     );
 };
```

Example: Using a regexp-parser()

In the following example, the incoming log message is the following:

```
Apr 20 11:09:46 test_field -> test_value
```

The regexp-parser inserts the `.regexp.` 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 parser and the destination.

```
source s_network {
     network(
         port(21514)
         flags(no-parse)
     );
 };
 parser p_regexp {
     regexp-parser(
         patterns(".*test_field -> (?<test_field>.*)$")
         prefix(".regexp.")
     );
 };
 destination d_file {
     file(
         "/tmp/test.json"
         template("$(format-json --scope dot-nv-pairs)\n")
     );
 };
 log {
     source(s_network);
     parser(p_regexp);
     destination(d_file);
 };
```

You can also define the parser inline in the log path.

```
   source s_network {
        network(
            port(21514)
            flags(no-parse)
        );
    };
    destination d_file {
        file(
            "/tmp/test.json"
            template("$(format-json --scope dot-nv-pairs)\n")
        );
    };
    log {
        source(s_network);
        parser{
            regexp-parser(
                patterns(".*test_field -> (?<test_field>.*)$")
                prefix(".regexp.")
            );
        };
        destination(d_file);
    };
```

You can set multiple patterns:

```
   parser p_regexp {
        regexp-parser(
            patterns(".*test_field -> (?<test_field>.*)$", ".*other_format: (?
<foo>.*)$")
            prefix(".regexp.")
        );
    };
```

14.21.1 Options of Regular expression parsers

The Regular expression parser has the following options.

flags()

Synopsis:	flags(“ignore-case” “unicode”)
Mandatory:	no

Description: The flags to apply for the parser. The available flags depend on the `type()` of the regular expression. For details, see Options of regular expressions.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

patterns()

Synopsis:	patterns(“pattern1” “pattern2”)
Mandatory:	yes

Description: The regular expression patterns that you want to find a match. `regexp-parser()` supports multiple patterns, and stops the processing at the first successful match.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

This parser does not have a default prefix. To configure a custom prefix, use the following format:

```
parser p_regexp{
     regexp-parser(
         patterns( ... )
         prefix("myprefix.")
     );
 };
```

template()

Synopsis: `template("${<macroname>}")`

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, the parser processes the entire message (`${MESSAGE}`).

type()

Synopsis:	`pcre` , `string` , `glob`
Mandatory:	no, defaults to `pcre`

Description: Sets how the `patterns()` expressions are interpreted: as Perl Compatible Regular Expressions (`pcre` , used by default), literal string searches (`string`), or glob patterns without regular expression support (`glob`).

14.22 Structured data (SDATA) parser

Available in AxoSyslog 4.1 and later.

The `sdata-parser()` allows you to parse an RFC5424-style structured data string. You can use it to parse this relatively complex format separately, for example, to process malformatted messages. You can use the optional `prefix` option to add a specific string before the names of the parsed name-value pairs.

See also the related `get_sdata()` FilterX function, which extracts the SDATA of the current message into a dict. Unlike `sdata-parser()`, it cannot parse an arbitrary string, and has no `prefix` option.

Declaration

```
parser parser_name {
     sdata-parser(
         template("<string-or-template-to-parse>")
         prefix("<prefix-for-parsed-name-value-pairs>")
     );
 };
```

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

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}`).

14.23 Sudo parser

The sudo parser can parse the log messages of the sudo command.

Prerequisites

- AxoSyslog version 4.16 or later.
- Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

  Your configuration must also contain `@include "scl.conf"` , which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

  The `sudo-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/sudo/sudo.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { system(); };
     parser { sudo-parser(); };
     destination { ... };
 };
```

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `sudo-parser()` uses the `.sudo.` prefix. To modify it, use the following format:

```
parser {
     sudo-parser(prefix("myprefix."));
 };
```

14.24 Parsing syslog messages

By default, AxoSyslog 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 AxoSyslog, 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**
>
> **AxoSyslog has several parsers that you can use to parse non-compliant messages. You can even write a custom 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));
```

Parsing errors

AxoSyslog 4.7 and newer automatically adds the following tags if it encounters errors when parsing syslog messages.

- `message.utf8_sanitized` : The message is not valid UTF-8.
- `syslog.missing_timestamp` : The message has no timestamp.
- `syslog.invalid_hostname` : The hostname field doesn't seem to be valid, for example, it contains invalid characters. For details on the valid characters, see the `check-hostname()` global option .
- `syslog.missing_pri` : The priority (PRI) field is missing from the message.
- `syslog.unexpected_framing` : An octet count was found in front of the message, suggested invalid framing.
- `syslog.rfc3164_missing_header` : The date and the host are missing from an RFC3164-formatted message - practically that's the entire header of RFC3164-formatted messages.
- `syslog.rfc5424_unquoted_sdata_value` : An RFC5424 message contains an incorrectly quoted SDATA field.
- `message.parse_error` : Some other parsing error occurred.

14.24.1 Options of syslog-parser() parsers

The `syslog-parser()` has the following options:

default-facility()

Type:	facility string
Default:	user

Description: This parameter assigns a facility value to the messages received from the source if the message does not specify one. The default is `user` (`kern` for kernel sources).

default-priority()

Type:	priority string
Default:	notice

Description: This parameter assigns a severity to the messages received from the source if the message does not specify one. For example, `default-priority(warning)` .

default-severity()

Type:	severity string
Default:	`notice`

Description: Assigns a default severity to messages received from this source when the message doesn't specify one. Accepts the standard syslog severity keywords (for example, `emerg` , `alert` , `crit` , `err` , `warning` , `notice` , `info` , `debug`). Combined with `default-facility()` to compute the message priority.

The `default-level()` and `default-priority()` options are aliases of `default-severity()` : all three set the severity part of the message priority. If you set more than one of them, the last one takes effect.

drop-invalid()

Type:	yes or no
Values:	`yes` or `no`
Default:	no

Description: This option determines how the `syslog-parser()` affects messages when parsing fails.

If you set `drop-invalid()` to `yes`, `syslog-parser()` will drop the message if the parsing fails.

If you set `drop-invalid()` to `no`, the parsing error triggers `syslog-parser()` to rewrite and extend the original log message with the following additional information:

- It prepends the following message to the contents of the `$MESSAGE` field: `Error processing log message`.
- It sets the contents of the `$PROGRAM` field to `syslog-ng`.
- It sets the contents of the `facility` field to `syslog`.
- It sets the contents of the `severity` field to `error`.

> **Note**
>
> **With the `drop-invalid(no)` option `syslog-parser()` will work in the same way as the sources which receive syslog-protocol/BSD-format messages.**

Example: enabling the drop-invalid() option

```
parser p_syslog {  syslog-parser(drop-invalid(yes)); };
```

flags()

Type:	`assume-utf8`, `check-hostname`, `check-program`, `dont-store-legacy-msghdr`, `empty-lines`, `exit-on-eof`, `expect-hostname`, `guess-timezone`, `ignore-aux-data`, `kernel`, `no-header`, `no-hostname`, `no-multi-line`, `no-parse`, `sanitize-utf8`, `store-legacy-msghdr`, `store-raw-message`, `syslog-protocol`, `validate-utf8`

Default: empty set

Description: Specifies the log parsing options of the source. For example:

```
source { network(flags(check-hostname, check-program)); };
```

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.

check-hostname

When receiving messages, AxoSyslog can check whether the hostname contains valid characters.

Valid characters are:

- alphanumeric characters (A-Z, a-z, 0-9)
- the dash (`-`) and underscore (`_`) characters
- the dot (`.`) and the colon (`:`) characters
- the `@` and slash (`/`)

If the hostname contains invalid characters, AxoSyslog sets the `syslog.invalid_hostname` tag for the message, and doesn't parse the `${HOST}` field from the message.

check-program

Available in version 4.10 and later.

If the `check-program` flag is enabled, AxoSyslog validates the `${PROGRAM}` field for RFC3164-formatted messages. Valid program names meet the following criteria:

- Contain only these characters: `[a-zA-Z0-9-_/().]`
- Include at least one alphabetical character.

If the program name fails validation, it's considered to be part of the log message.

You can also enable this behavior globally using the `check-program` global option .

dont-store-legacy-msghdr

By default, AxoSyslog 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 (AxoSyslog automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). If you do not want to store the original header of the message, enable the `dont-store-legacy-msghdr` flag.

empty-lines

Use the `empty-lines` flag to keep the empty lines of the messages. By default, AxoSyslog removes empty lines automatically.

exit-on-eof

If this flag is set on a source, AxoSyslog stops when an EOF (end of file) is received. Available in version 4.9 and later.

expect-hostname

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

guess-timezone

Attempt to guess the timezone of the message if this information is not available in the message. Works when the incoming message stream is close to real time, and the timezone information is missing from the timestamp.

ignore-aux-data

The `ignore-aux-data` flag stops AxoSyslog from collecting the auxiliary data that the transport layer can attach to each incoming message, and from setting the related name-value pairs. Auxiliary data includes:

- The UNIX credentials of the sender process on UNIX domain sockets: `${.unix.pid}`, `${.unix.uid}`, `${.unix.gid}`, `${.unix.cmdline}`, `${.unix.exe}`, `${.audit.auid}`, and `${.audit.ses}`. For details, see UNIX credentials and other metadata.

- The fields of the peer certificate on TLS-encrypted transports: `${.tls.x509_cn}`, `${.tls.x509_o}`, `${.tls.x509_ou}`, and `${.tls.x509_fp}`.
- The receive timestamp and the peer address that the transport layer provides for the message.

Use this flag if you do not need this metadata and want to avoid the cost of collecting it. Gathering UNIX credentials is especially expensive, because AxoSyslog reads the `/proc` filesystem for every message.

kernel

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

no-header

The `no-header` flag triggers AxoSyslog to parse only the `PRI` field of incoming messages, and put the rest of the message contents into `$MSG`.

Its functionality is similar to that of the `no-parse` flag, except the `no-header` flag does not skip the `PRI` field.

> **Note**
>
> **Essentially, the `no-header` flag signals AxoSyslog that the `syslog` header is not present (or does not adhere to the conventions / RFCs), so the entire message (except from the `PRI` field) is put into `$MSG`.**

The following example illustrates using the `no-header` flag with the `syslog-parser()` parser:

```
parser p_syslog {
    syslog-parser(
    flags(no-header)
    );
};
```

no-hostname

Enable the `no-hostname` flag if the log message does not include the hostname of the sender host. That way AxoSyslog 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, AxoSyslog 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 AxoSyslog 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, AxoSyslog generates a new syslog header (timestamp, host, and so on) automatically. Note that even though `flags(no-parse)` disables message parsing, some flags can still be used, for example, the `no-multi-line` flag.

sanitize-utf8

When using the `sanitize-utf8` flag, AxoSyslog converts non-UTF-8 input to an escaped form, which is valid UTF-8.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

store-legacy-msghdr

By default, AxoSyslog stores the original incoming header of the log message, so this flag is active. To disable it, use the `dont-store-legacy-msghdr` flag.

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 AxoSyslog 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.

Prior to version 4.6, this flag worked only when parsing RFC3164 messages. Starting with version 4.6, it works also for RFC5424 and raw messages.

For RFC5424-formatted messages, if the BOM character is missing, but the message is otherwise UTF-8 compliant, AxoSyslog automatically adds the BOM character to the message.

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

For the `syslog-parser()` you can also set the following flags:

- `check-hostname` : Equivalent with the `check-hostname()` global option , but only applies to this parser.
- `no-piggyback-errors` : Do not attribute the message to AxoSyslog in case of errors. Things already processed or extracted are retained, for example: `${MESSAGE}` retains its value (potentially the raw message), other macros like `${HOST}` , `${PROGRAM}` , or

`${PID}` may or may not be extracted. The error is indicated by setting `${MSGFORMAT}` set to “syslog:error”.

Available in AxoSyslog 4.8.1 and later.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

sdata-prefix()

Type:	string
Default:	`.SDATA.`

Available in AxoSyslog 4.1 and later.

Description: Adds a specific string before the names of the parsed SDATA fields to store the name-value pairs created from the SDATA fields separately. Note that unless the value of `sdata-prefix` starts with `.SDATA.` , using this option excludes the parsed fields from the `sdata` and `rfc5424` scopes of the value pairs .

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-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.

14.25 Parsing tags

The AxoSyslog application can tag log messages, and can include these tags in the log messages, as described in Tagging messages. The `tags-parser()` can parse these tags from the incoming messages and re-tag them. That way if you add tags to a log message on a AxoSyslog client, the message will have the same tags on the AxoSyslog server. Available in version 3.23 and later.

Specify the macro that contains the list of tags to parse in the `template()` option of the parser, for example, the SDATA field that you used to transfer the tags, or the name of the JSON field that contains the tags after using the `json-parser()`.

```
tags-parser(template("${<macro-or-field-with-tags>}"));
```

Options

The `tags-parser()` parser has the following options.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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}`).

14.26 Websense parser

The Websense parser can parse the log messages of Websense Content Gateway (Raytheon|Websense, now Forcepoint). These messages do not completely comply with the syslog RFCs, making them difficult to parse. The `websense-parser()` of AxoSyslog solves this problem, and can separate these log messages to name-value pairs. For details on using value-pairs in AxoSyslog see Structuring macros, metadata, and other value-pairs. The parser can parse messages in the following format:

```
<PRI><DATE> <TIMEZONE> <IP-ADDRESS> <NAME=VALUE PAIRS>
```

For example:

```
<159>Dec 19 10:48:57 EST 192.168.1.1 vendor=Websense product=Security
product_version=7.7.0 action=permitted severity=1 category=153 user=-
src_host=192.168.2.1 src_port=62189 dst_host=example.com dst_ip=192.168.3.1
dst_port=443 bytes_out=197 bytes_in=76 http_response=200 http_method=CONNECT
http_content_type=-
http_user_agent=Mozilla/5.0_(Windows;_U;_Windows_NT_6.1;_enUS;_rv:1.9.2.23)_Gecko/2
0110920_Firefox/3.6.23 http_proxy_status_code=200 reason=- disposition=1034
policy=- role=8 duration=0 url=https://example.com
```

If you find a message that the `websense-parser()` cannot properly parse, contact us , so we can improve the parser.

The AxoSyslog application sets the `${PROGRAM}` field to `Websense` .

By default, the websense-specific fields are extracted into name-value pairs prefixed with `.websense` . For example, the `product_version` in the previous message becomes `${.websense.product_version}` . You can change the prefix using the `prefix` option of the parser.

Prerequisites

Install the `axosyslog-scl` package on Debian/Ubuntu . On RHEL and compatible distributions , the SCL files are part of the `axosyslog` base package.

Your configuration must also contain `@include "scl.conf"`, which the default configuration file already does. If the SCL files are missing, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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

The `websense-parser()` driver is a reusable configuration snippet (SCL). For details on using or writing such configuration snippets, see Reusing configuration blocks. You can find its source in scl/websense/plugin.conf on GitHub.

Declaration:

```
@version: 4.29
 @include "scl.conf"
 log {
     source { network(flags(no-parse)); };
     parser { websense-parser(); };
     destination { ... };
 };
```

Note that you have to disable message parsing in the source using the `flags(no-parse)` option for the parser to work.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

By default, `websense-parser()` uses the `.websense.` prefix. To modify it, use the following format:

```
parser {
     websense-parser(prefix("myprefix."));
 };
```

14.27 Windows XML Event Log (EVTX) parser

The `windows-eventlog-xml-parser()` can parse messages in the Windows XML Event Log (EVTX) format.

Prerequisites

Available in AxoSyslog version 4.5 and later.

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Configuration

See also the equivalent FilterX function, `parse_windows_eventlog_xml()` .

Example configuration:

```
parser p_win {
    windows-eventlog-xml-parser(prefix(".winlog."));
};
```

The `windows-eventlog-xml-parser()` parser has the same parameters are the same as the `xml()` parser .

Don't forget to include the parsers in a log statement to actually use it:

```
log {
    source(s_local);
    parser(windows-eventlog-xml-parser(prefix(".winlog.")));
    destination(d_local);
};
```

14.28 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.

See also the related FilterX function `parse_xml()` . It is a separate implementation that parses XML into a dict, so the options and limitations of this parser do not apply to it.

Prerequisites

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-xml-parser` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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.

> **Note**
>
> **If your log messages are entirely in .xml format, make sure to disable any message parsing on the source side by including the `flags("no-parse")` option in your source statement. This will put the entire log message in the `$MESSAGE` macro, which is the field that the XML parser parses by default.**

14.28.1 Limitations of the XML parsers

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, AxoSyslog 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.

14.28.2 Options of the XML parsers

The XML parser has the following options.

create-lists()

Accepted values:	`yes` , `no`
Default:	`yes`
Mandatory:	no

Available in AxoSyslog version 3.20 and later.

Description: If an XML element appears multiple times on the same level, the parser stores the repeated values as a list-typed name-value pair. If you set `create-lists(no)` , the parser concatenates the repeated values into a single string instead, which was the behavior before version 3.20.

For example, from the following XML input:

```
<tag><item>first</item><item>second</item></tag>
```

The parser creates the `${.xml.tag.item}` name-value pair with the list value `first,second` . With `create-lists(no)` , the value of `${.xml.tag.item}` is the string `firstsecond` .

drop-invalid()

Synopsis:	drop-invalid()
Format:	`yes` or `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>
</tag3>
```

The following output is generated:

```
    {"_xml":{"tag3":"Text3"}}
```

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

The `prefix()` option is optional and its default value is `".xml"` .

strip-whitespaces()

Synopsis:	strip-whitespaces()
Format:	`yes` or `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}`).

15 Correlating log messages

Group related log messages and correlate them into a single context with the grouping-by() parser.

The AxoSyslog 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 `group-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 email server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message. Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages with AxoSyslog, 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.

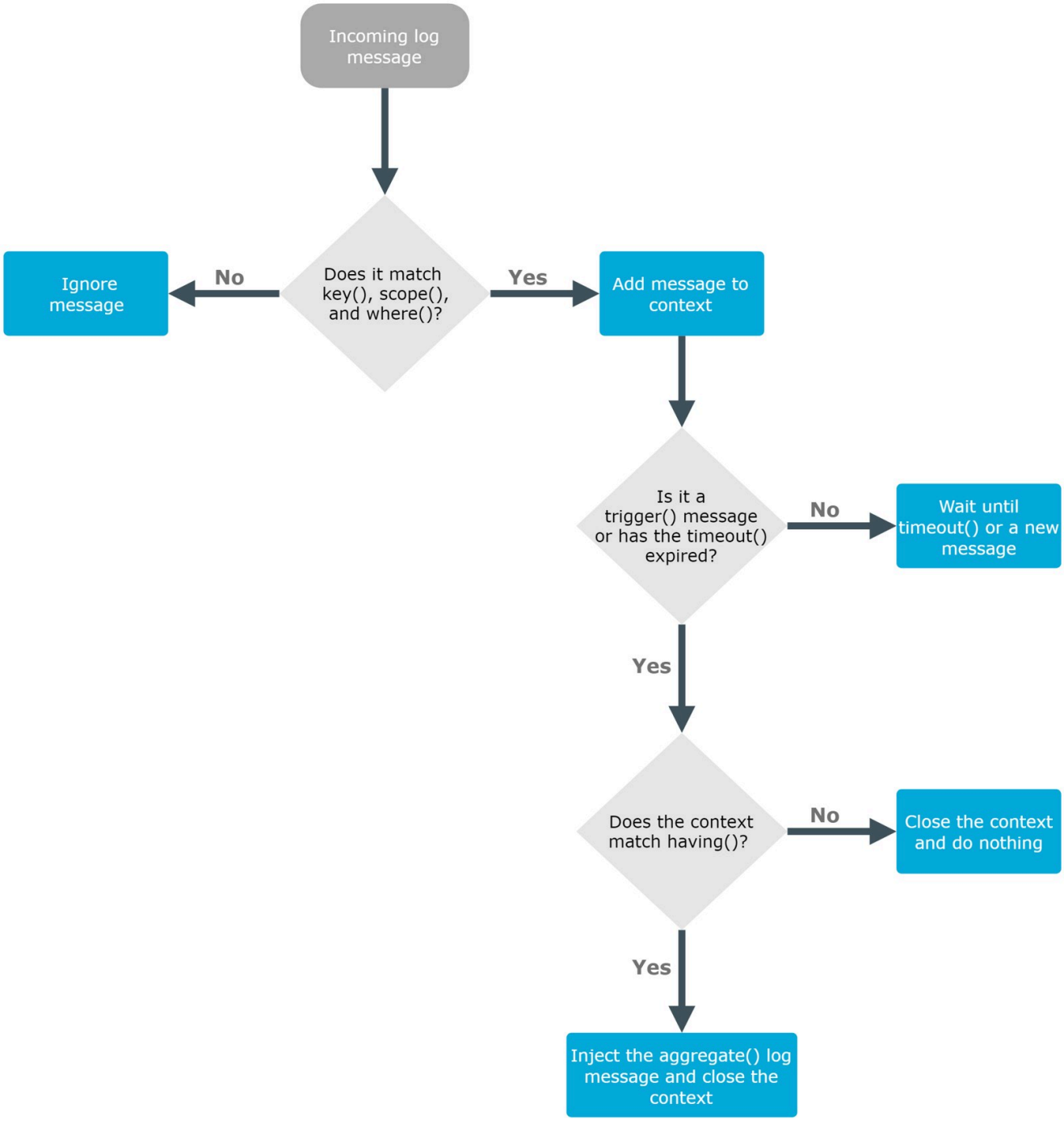
15.1 Correlating messages using the grouping-by() parser

The AxoSyslog application can correlate log messages that match a set of filters. This works similarly to SQL GROUP BY statements. Alternatively, you can also correlate log messages using pattern databases. For details, see Correlating log messages using pattern databases.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix email server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message. Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages with AxoSyslog, 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 AxoSyslog 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), AxoSyslog generates and sends the message set in the `aggregate()` option.

> **Note**
>
> **Message contexts are persistent and are not lost when AxoSyslog is reloaded (SIGHUP), but are lost when AxoSyslog 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, AxoSyslog will restart the timeout using the `context-timeout` set for the new message.
- When calculating if the timeout has already expired or not, AxoSyslog 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 AxoSyslog 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), AxoSyslog will replace its timestamp with the current system time.

Example: How AxoSyslog 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 AxoSyslog receives the messages within 1 second, the timeout event will occur immediately, because the difference of the two timestamp (60 seconds) is larger than the timeout value (10 seconds).

- 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 AxoSyslog, 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)
     );
 };
```

15.1.1 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}` .

Warning

When referencing an earlier message of the context, always enclose the field name between braces, for example, `${PID}@3` . The reference will not work if you omit the braces.

Note

To use a literal `@` character in a template, use `@@` .

Example: Referencing values from an earlier message

The following action can be used to log the length of an SSH session (the time difference between a login and a logout message in the context):

```
   aggregate(
        value('value name="MESSAGE" An SSH session for ${SSH_USERNAME}@1 from
${SSH_CLIENT_ADDRESS}@2 closed. Session lasted from ${DATE}@2 to ${DATE}')
    )
```

If you do not know 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 .

15.1.2 Options of grouping-by parsers

The `grouping-by` has the following options.

aggregate()

Synopsis: aggregate()

Description: Specifies the message that AxoSyslog 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` : AxoSyslog 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: AxoSyslog 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 AxoSyslog 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.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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.**

persist-name()

Type: string

Default: N/A

Description: If you receive the following error message during AxoSyslog 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("<prefix>")`

Description: Inserts a prefix before the name part of the name-value pairs that the `aggregate()` option adds to the generated message. For example, with `prefix(".grouping-by.")` the name-value pair set by `value("count" "...")` becomes `${.grouping-by.count}`.

scope()

Type: `process`, `program`, `host`, or `global`

Default: `global`

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.

sort-key()

Synopsis: sort-key()

Description: Allows sorting of the elements before they are aggregated into a context. Use this when entries are not received in order. This works similarly to the SQL ORDER BY keyword.

> **Note**
>
> - Sorting is done by AxoSyslog when the context is about to be closed by trigger() or timeout(), but before AxoSyslog evaluates the having() option.
> - AxoSyslog can slow down if you specify several sort-key macro or template options, for example, `sort-key("${3}${4}")` .

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 AxoSyslog 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, AxoSyslog generates and sends the triggered message (specified in the aggregate() option), and clears the context.

where()

Synopsis: where()

Description: Specifies a filter condition. Messages not matching the filter will not be added to the context. Note that the `where()` filter has access only to the current message.

16 Enriching log messages with external data

Add data to log messages from external CSV files, from GeoIP2 lookups of IP addresses, or with custom Python code.

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 AxoSyslog 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 AxoSyslog 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 Python parser.

16.1 Adding metadata from an external file

In AxoSyslog 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, for example, the hostname. To use shell-style globbing (wildcards) in selectors, see Shell-style globbing in the selector. You can also reference the name of a filter that matches the messages, see Using filters as selector
- The name of the name-value pair that AxoSyslog adds to matching log messages.
- The value of the name-value pairs. Starting with AxoSyslog version 3.22, the value of the name-value pair can be a template or a template function, for example, `"selector3,name,$(echo $HOST_FROM)";`

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
```

Prerequisites

On Debian/Ubuntu , this feature is available in a separate module: install the `axosyslog-mod-add-contextual-data` package. On RHEL and compatible distributions , it's part of the `axosyslog` base package. If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

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 AxoSyslog 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 AxoSyslog for the changes to take effect. If reloading AxoSyslog or the database file fails for some reason, AxoSyslog 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
```

16.1.1 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, AxoSyslog evaluates the filters in the order they appear in the database file. If a filter matches the message, AxoSyslog 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 AxoSyslog 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, AxoSyslog 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 AxoSyslog 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: 4.29
 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 AxoSyslog for the changes to take effect. If reloading AxoSyslog or the files fails for some reason, AxoSyslog will keep using the last working version of the file.

16.1.2 Shell-style globbing in the selector

Starting with in AxoSyslog 3.24 and later, you can use shell-style globbing ('*' and '?' wildcards) in the selector.

To use globs in a selector

1. Use the `glob()` option within the `selector()` option in your AxoSyslog configuration file, for example:

```
parser p_add_context_data {
    add-contextual-data(
        selector(glob("${HOST}"))
        database("context-info-db.csv")
    );
};
```

2. Use globs and wildcards in the selector column of your CSV-file, for example:

```
example-glob-entry1*,sourcetype,:hec:user
example-glob-entry2*,sourcetype,:hec:user
postfix*,sourcetype,:hec:mta
```

Note the following points when using globbing in the selector:

- The order of the patterns depends on the CSV-file. The order of entries in the database determines the matching order.
- The globs are matched against the expanded template string sequentially.
- Put more specific patterns to the top of the CSV-file. The AxoSyslog appication does not evaluate other entries after the first match.
- In debug mode, AxoSyslog sends log messages to its `internal()` destination to help troubleshooting. For example:

```
    [2019-09-21T06:01:10.748237] add-contextual-data(): Evaluating glob against
message; glob-template='$PROGRAM', string='postfix/smtpd', pattern='example-
glob-entry1*', matched='0'
    [2019-09-21T06:01:10.748562] add-contextual-data(): Evaluating glob against
message; glob-template='$PROGRAM', string='postfix/smtpd', pattern='example-
glob-entry2*', matched='0'
    [2019-09-21T06:01:10.748697] add-contextual-data(): Evaluating glob against
message; glob-template='$PROGRAM', string='postfix/smtpd', pattern='postfix*',
matched='1'
    [2019-09-21T06:01:10.750084] add-contextual-data(): message lookup
finished; message='almafa', resolved_selector='postfix*', selector='postfix*',
msg='0x8e15320'
```

16.1.3 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` 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 AxoSyslog 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 AxoSyslog evaluates for each message, and if its value matches the ID of an entry in the database, AxoSyslog adds the name-value pair of every matching database entry to the log message. You can use the following in the `selector()` option.

- Strings
- A single macro (for example, `selector("${HOST}")`)
- To use filters as selectors, see `selector(filters())`.
- To use shell-style globbing (wildcards) in selectors, see `selector(glob())`.

- Using templates as selectors is not supported.

selector(filters())

Synopsis: selector(filters())

Description: Use filters as selectors to look up entries in the database. For details and examples, see Using filters as selector.

selector(glob())

Synopsis: selector(glob())

Description: Use shell-style globbing (wildcards) in selectors to look up entries in the database. For details and examples, see Shell-style globbing in the selector.

16.2 Looking up GeoIP data from IP addresses (DEPRECATED)

This parser is deprecated. Use Looking up GeoIP2 data from IP addresses instead.

The AxoSyslog 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 GeoLite2City 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, AxoSyslog 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

    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>an 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":"prg
00000","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 elasticsearch2: DEPRECATED - Send messages directly to Elasticsearch version 2.0 or higher.

```
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"
                   }
                }
             }
          }
       }
    }
 }
```

16.2.1 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

For example, to insert the `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 AxoSyslog must have the required privileges to read this file. Do not modify or delete this file while AxoSyslog is running, it can crash AxoSyslog.

16.3 Looking up GeoIP2 data from IP addresses

The AxoSyslog 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 AxoSyslog application works with the Country and the City version of the GeoIP2 database, both free and the commercial editions. The AxoSyslog application works with the `mmdb` (GeoIP2) format of these databases. Other formats, like `csv` are not supported.

Prerequisites

This feature requires a separate module. Install the `axosyslog-mod-geoip2` package on Debian/Ubuntu , or the `axosyslog-geoip` package on RHEL and compatible distributions . If the module isn't installed, AxoSyslog fails to start with an `unexpected LL_IDENTIFIER` error .

Download the GeoIP2 database

> **Note**
>
> To access longitude and latitude information, download the City version of the GeoIP2 database.
>
> There are two types of GeoIP2 databases available.
>
> - *GeoLite2 City:*
>   - free of charge
>   - less accurate
> - *GeoIP2 City:*
>   - has to be purchased
>   - more accurate
>
> Unzip the downloaded database (for example, to the `/usr/share/GeoIP2/GeoIP2City.mmdb` file). This path will be used later in the configuration.
>
> Starting with version 3.24, AxoSyslog tries to automatically detect the location of the database. If that is successful, the `database()` option is not mandatory.

16.3.1 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 check the GeoIP Databases of MaxMind. However, note that the AxoSyslog application works with the `mmdb` (GeoIP2) format of these databases. Other formats, like `csv` are not supported.

16.3.2 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, AxoSyslog 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

 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","accu
racy_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"}
```

16.3.3 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: DEPRECATED - Send messages directly to Elasticsearch version 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"
                   }
                }
             }
          }
       }
    }
 }
```

16.3.4 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 AxoSyslog. If you use such a macro name as the name of a parsed value, it will attempt to replace the original value of the macro (note that only soft macros can be overwritten, see Hard versus soft macros for details). To avoid such problems, use a prefix when naming the parsed values, for example, `prefix(my-parsed-data.)`

For example, to insert the `.geoip2` prefix, use the `prefix(.geoip2)` option. To refer to a particular data when using a prefix, use the prefix in the name of the macro, for example, `${geoip2.country_code}` .

database()

Synopsis: database()

Default:

Description: Path to the GeoIP2 database to use. This works with absolute and relative paths as well. Note that AxoSyslog must have the required privileges to read this file. Do not modify or delete this file while AxoSyslog is running, it can crash AxoSyslog.

Starting with version 3.24, AxoSyslog tries to automatically detect the location of the database. If that is successful, the `database()` option is not mandatory.

internal()

Accepted values:	`yes` , `no`
Default:	`no`

Description: Marks this pipeline element as internal. Elements marked as `internal()` are treated as an implementation detail, so for example statistics of the given pipe are available only on higher stats level. This option is mainly useful for developers or when writing SCL blocks and integrations.

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}`).

17 Statistics and metrics of AxoSyslog

Monitor AxoSyslog with metrics and statistics: query them with syslog-ng-ctl, read them from the internal() source, or scrape them with Prometheus.

The AxoSyslog application collects various statistics and metrics about its performance and status for observability and monitoring. Which metrics and statistics are collected depends on the configuration of AxoSyslog and the value of the `stats(level())` global option .

Metrics and statistics

- AxoSyslog provides detailed metrics about its performance and status for observability and monitoring. We recommend using Prometheus to scrape these metrics, see Collect metrics with Prometheus for details. To display the current metrics locally in Prometheus-compatible format, run:

  ```
  syslog-ng-ctl stats prometheus
  ```

  Note that which metrics are shown depends on the current value of the `stats(level())` global option (you can list the available metrics by running `syslog-ng --metrics-registry`). For details on what the metrics mean, see Metrics reference.

- Statistics are a legacy way to access the status of AxoSyslog. Metrics are newer and in active development. Many metrics aren't available as legacy statistics.

  You can access legacy statistics using the following methods.

  - The `syslog-ng-ctl query` command gives structured access to the selected legacy statistics..
  - The `syslog-ng-ctl stats` command lists all the available legacy statistics in bulk.
  - Using the `internal()` source . We recommend using either of the previous two methods instead.

  For details about the available counters and the output format, see Statistics reference.

17.1 Collect metrics with Prometheus

Export AxoSyslog and syslog-ng metrics to Prometheus using the `axosyslog-metrics-exporter` and scrape them with Prometheus.

Prerequisites

- A running AxoSyslog instance
- `stats(level(2))` or higher set in your configuration file
- File-level access to the AxoSyslog control socket

> **Note**
>
> **You must set `stats(level(2))` to expose host-level metrics. Without it, many metrics (including per-host counters) aren't available. For details, see the `stats(level())` global option .**

Deploy the metrics exporter

The `axosyslog-metrics-exporter` is a Go-based tool that exposes Prometheus-style metrics by connecting to the AxoSyslog control socket. It works with syslog-ng, syslog-ng Premium Edition, and all versions of AxoSyslog (syslog-ng™ is the trademark of One Identity LLC).

Run the exporter as a container:

```
sudo podman run -d -p 9577:9577 -v $(echo /var/*/syslog-ng/syslog-ng.ctl):/syslog-ng.ctl \
  ghcr.io/axoflow/axosyslog-metrics-exporter:latest --socket.path=/syslog-ng.ctl
```

Once started, the metrics endpoint is available at `http://127.0.0.1:9577/metrics`.

> **Note**
>
> **The control socket is typically located at `/var/lib/syslog-ng/syslog-ng.ctl` or `/var/run/syslog-ng/syslog-ng.ctl`. In containerized environments, share the Unix domain socket with the exporter container using a volume mount, as shown in the preceding command.**

Configure Prometheus

Create a `prometheus.yml` file with a scrape job pointing to the metrics exporter:

```
scrape_configs:
  - job_name: axosyslog
    static_configs:
      - targets:
          - <prometheus-host-ip>:9577
        labels:
          app: axosyslog
```

Then run Prometheus:

```
sudo podman run \
    -p 9090:9090 \
    -v ./prometheus.yml:/etc/prometheus/prometheus.yml \
    prom/prometheus
```

To verify that Prometheus is scraping correctly, open the following pages in your browser:

- `http://127.0.0.1:9090/config` : shows the active configuration
- `http://127.0.0.1:9090/targets` : shows whether the AxoSyslog scrape target is up

Key metrics to monitor

For a detailed reference, see Metrics reference. The main metrics that you should monitor are the following.

Critical metrics

These metrics indicate problems that require immediate attention:

- `output_unreachable` : destination is unavailable
- `socket_receive_dropped_packets_total` : messages dropped on the source side
- `output_events_total{result="dropped"}` : messages dropped at the output without flow control

- `socket_rejected_connections_total` : number of rejected incoming connections

Core pipeline metrics

These metrics give you a basic understanding of pipeline throughput:

- `input_events_total` : total messages received by all sources
- `output_events_total` : total messages sent by all destinations
- `filtered_events_total` : total messages processed by filters
- `parsed_events_total` : total messages processed by parsers
- `memory_queue_events` and `disk_queue_events` : current buffer usage
- `io_worker_latency_seconds` : I/O worker latency, a sign of potential overload

17.2 Metrics reference

The following list shows the metrics available in AxoSyslog.

AxoSyslog provides detailed metrics about its performance and status for observability and monitoring. We recommend using Prometheus to scrape these metrics, see Collect metrics with Prometheus for details. To display the current metrics locally in Prometheus-compatible format, run:

```
syslog-ng-ctl stats prometheus
```

Note that which metrics are shown depends on the current value of the `stats(level())` global option (you can list the available metrics by running `syslog-ng --metrics-registry`). For details on what the metrics mean, see Metrics reference.

> **Note**
>
> - Metrics that have the `_total` suffix reset to zero when AxoSyslog is restarted. Reloading AxoSyslog doesn't cause a reset.
> - Different metrics are available on different `stats(level())` .

classified_events_total

Description: Default metric of the `metrics-probe()` parser.

disk_queue_capacity_bytes

Description: Maximal size of the disk queue (in bytes), as set in the `capacity-bytes()` disk-buffer option.

disk_queue_capacity

Description: The size of the overflow queue of the destination, as set in the `flow-control-window-size()` disk-buffer option.

disk_queue_dir_available_bytes

Description: The size of the space available in the directories where disk-buffer files are stored (including directories storing abandoned disk-buffers), in bytes.

disk_queue_disk_allocated_bytes

Description: The actual size of the disk-buffer files, in bytes.

disk_queue_disk_usage_bytes

Description: Total size of data waiting in each disk-buffer, in bytes.

disk_queue_events

Description: Number of messages waiting in each disk-buffer by destination.

disk_queue_memory_usage_bytes

Description: Amount of memory used for caching disk-buffers, in bytes.

disk_queue_processed_events_total

Description: The number of events processed since startup by each disk-buffer.

event_processing_latency_seconds

Description: Histogram of the latency (time from receiving the message to fully processing it), from the source or destination perspective.

events_allocated_bytes

Description: The total amount of memory used by log messages in AxoSyslog.

filtered_events_total

Description: The total number of messages that matched and didn't match a filter, for each filter in the configuration file.

input_event_bytes_total

Description: Incoming log messages processed by each source, measured in bytes.

input_events_total

Description: Number of incoming log messages processed by each source.

input_transport_errors_total

Description: Number of various transport errors that prevent AxoSyslog from ingesting messages, for example, TLS handshake errors or syslog framing errors. Labels include the source `id`, `peer_address`, and the `reason` for the error.

```
syslogng_input_transport_errors_total{address="127.0.0.1:5513",driver="syslog",peer
_address="127.0.0.1",reason="invalid-frame-header",transport="tcp"} 1
syslogng_input_transport_errors_total{address="127.0.0.1:5515",driver="syslog",id="
s_tls_req#0",peer_address="127.0.0.1",reason="tls-
handshake",tls_error="0A0000C7",tls_error_string="SSL routines::peer did not return
a certificate",transport="tls"} 1
```

input_window_available

Description: Available on `stats(level(3))`. Shows the current size of the flow-control window (how much is still free from `log-iw-size()`).

input_window_capacity

Description: Available on `stats(level(3))`. Shows the value of `log-iw-size()` (the size of the flow-control window).

input_window_full_total

Description: The total number of input window full events, for the entire configuration. These events cause AxoSyslog to throttle the source. Available on `stats(level(1))`.

internal_events_queue_capacity

Description: The internal queue size of the `internal()` source.

internal_events_total

Description: The number of messages the `internal()` source has queued, processed, or dropped.

io_worker_latency_seconds

Description: Shows how overloaded the IO workers of AxoSyslog are.

last_config_file_modification_timestamp_second

Description: The date when the configuration file was last modified.

last_config_reload_timestamp_seconds

Description: The date when the AxoSyslog configuration was last reloaded. If it differs from `last_successful_config_reload_timestamp_seconds`, reloading the configuration has failed.

last_successful_config_reload_timestamp_secon

Description: The date when the AxoSyslog configuration was last reloaded successfully.

mainloop_io_worker_roundtrip_latency_seconds

Description: Shows how overloaded the main AxoSyslog loop is (how much time it takes to start a new worker). Values close to 0 are good, high values indicate high load or processing bottleneck.

memory_queue_capacity

Description: Shows the capacity (maximum possible size) of each queue. Note that this metric publishes `log-fifo-size()`, which only limits non-flow-controlled messages. Messages coming from flow-controlled paths aren't limited by `log-fifo-size()`, but by the `log-iw-size()` of their corresponding source. For metrics on `log-iw-size()`, see `input_window_available` and `input_window_capacity`.

memory_queue_events

Description: Number of messages waiting in each memory queue by destination.

memory_queue_memory_usage_bytes

Description: Total bytes of data waiting in each memory queue.

memory_queue_processed_events_total

Description: The number of events processed since startup by each queue.

output_active_worker_partitions

Description: The number of active partitions when `worker-partition-autoscaling()` is set to `yes`.

output_batch_size_bytes

Description: Histogram-style metrics for the destination.

output_batch_size_events

Description: Histogram-style metrics for the destination.

output_batch_timedout_total

Description: For destinations that use batching, it shows the number of batches that were sent because of timeout (either `batch-timeout()` or `batch-idle-timeout()` expired).

output_event_bytes_total

Description: Log messages sent to each destination, measured in bytes.

output_event_latency_seconds

Description: Histogram of the latency: time from receiving the message to delivering it to the destination.

output_event_retries_total

Description: Shows the number of events when AxoSyslog retried sending a message.

output_event_size_bytes

Description: Histogram-style metrics for the destination.

output_events_total

Description: Number of log messages sent to each destination, showing sent and dropped messages.

output_grpc_requests_total

Description: The total number of gRPC requests.

output_http_requests_total

Description: Available on `stats(level(1))` The total number of HTTP requests.

output_request_latency_seconds

Description: Histogram-style metrics for the destination.

output_unreachable

Description: A bool-like metric, which shows whether a destination is reachable or not.

output_workers

Description: The number of workers configured for each destination.

parallelize_failed_events_total

Description: The number of events that `parallelize()` couldn't process in parallel. Such messages were sent without parallelization. A high number of such events can signal a configuration issue or a bottleneck.

parallelized_assigned_events_total

Description: The number of events each worker has received when using `parallelize()` . Can show if the workers receive the load unevenly.

parallelized_batch_size

Available in AxoSyslog 4.26 and later at `stats-level(4)`

Description: Prometheus-style histograms to that show how logscheduler batches messages. Useful only for low-level debugging and troubleshooting.

```
syslogng_parallelized_batch_size_sum{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5"} 1000000
syslogng_parallelized_batch_size_count{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5"} 13962
syslogng_parallelized_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="1"} 4686
syslogng_parallelized_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="2"} 2857
syslogng_parallelized_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="4"} 2757
syslogng_parallelized_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="8"} 1401
```

parallelized_input_batch_size

Available in AxoSyslog 4.26 and later at `stats-level(4)`

Description: Prometheus-style histograms to that show how logscheduler batches messages. Useful only for low-level debugging and troubleshooting.

Example:

```
syslogng_parallelized_input_batch_size_sum{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5"} 1000000
syslogng_parallelized_input_batch_size_count{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5"} 11979
syslogng_parallelized_input_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="1"} 4545
syslogng_parallelized_input_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="2"} 2784
syslogng_parallelized_input_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="4"} 2729
syslogng_parallelized_input_batch_size_bucket{parallelize="/install/etc/callgrind-syslog-ng.conf:30:5",le="8"} 1429
```

parallelized_processed_events_total

Description: The number of events processed using `parallelize()` .

parsed_events_total

Description: Shows the number of messages processed by each parser.

route_egress_total

Description: The number of messages delivered by each named log path .

route_ingress_total

Description: The number of messages entering each named log path .

scratch_buffers_bytes

Description: The number of bytes allocated to internal string buffers.

scratch_buffers_count

Description: The number of allocated internal string buffers.

socket_connections

Description: Number of active connections for the sources.

socket_max_connections

Description: Maximum permitted number of connections for the sources.

socket_receive_buffer_max_bytes

Description: The maximal size socket receive buffer in bytes, as configured in the `so-rcvbuf()` option of the destination.

socket_receive_buffer_used_bytes

Description: The number of bytes used from the socket receive buffer.

socket_receive_dropped_packets_total

Description: Number of UDP packets dropped by the OS before processing.

socket_rejected_connections_total

Description: The number of connections rejected because the `max-connections()` limit of the source was reached, for each source.

stats_level

Description: Shows the current verbosity `level()` of statistics and metrics.

tagged_events_total

Description: The number of messages marked with a tag , for each tag. (Every message automatically has the tag of its source in `.source.<id_of_the_source_statement>` format.)

17.3 Statistics reference

Statistics are a legacy way to access the status of AxoSyslog. Metrics are newer and in active development. Many metrics aren't available as legacy statistics.

You can list all active statistics on your AxoSyslog host using the following command (this lists the statistics, without their current values): `syslog-ng-ctl query list "*"`

Format of statistics

To list the statistics and their values, use the following command: `syslog-ng-ctl query get "*"`

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 displayed statistics have the following structure.

- The type of the object (for example, `dst.file`, `tag`, `src.facility`)
- The ID of the object used in the `syslog-ng.conf` configuration file, for example, `d_internal` or `source.src_tcp`. The `#0` part means that this is the first destination in the destination group.
- 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.
- The status of the object. One of the following:
  - `a`: active. At the time of querying the statistics, the source or the destination was still alive (it continuously received statistical data).
  - `d`: dynamic. Such objects may not be continuously available, for example, like statistics based on the sender's hostname. These counters only appear above a certain value of `stats(level())` global option:

- `host` : source host, from `stats(level(2))`
- `program` : program, from `stats(level(3))`
- `sender` : sender host, from `stats(level(3))`

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 AxoSyslog, 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 AxoSyslog application stores the statistics of the objects when AxoSyslog is reloaded. However, if the configuration of AxoSyslog changed since the last reload, the statistics of orphaned objects are deleted.**

The `connections` statistics counter displays the number of connections tracked by AxoSyslog for the selected source driver.

Example configuration and statistics output

The following configuration will display the following `syslog-ng-ctl` statistics output:

Configuration:

```
source s_network {
  tcp(
    port(8001)
  );
};
```

Statistics output:

```
src.tcp;s_network#0;tcp,127.0.0.5;a;processed;1
src.tcp;s_network#0;tcp,127.0.0.1;a;processed;3
src.tcp;s_network;afsocket_sd.(stream,AF_INET(0.0.0.0:8001));a;connections;2
```

Statistics reference

The type of the statistics:

- `batch_size_avg` : When batching is enabled, then this shows the current average batch size of the given source or destination.
- `batch_size_max` : When batching is enabled, the value of `batch_size_max` shows the current largest batch size of the given source or destination.
- `discarded` : The number of messages discarded by the given parser. These are messages that the parser could not parsed, and are therefore not processed. For example:

  ```
  parser;demo_parser;;a;discarded;20
  ```

- `dropped` : The number of dropped messages. AxoSyslog could not send these messages to the destination and the output buffer got full, so messages were dropped by the destination driver, or AxoSyslog dropped the message for some other reason (for example, a parsing error).
- `eps_last_1h` : The EPS value of the past 1 hour.
- `eps_last_24h` : The EPS value of the past 24 hours.
- `eps_since_start` : The EPS value since the current AxoSyslog start.

> **Note**
>
> When using the `eps_last_1h`, the `eps_last_24h`, and the `eps_since_start` statistics, consider the following:
>
> - EPS stands for "event per second", and in our case, a message received or sent counts as a single event.
> - The `eps_last_1h`, the `eps_last_24h`, and the `eps_since_start` values are only approximate values.
> - The `eps_last_1h`, the `eps_last_24h`, and the `eps_since_start` values are automatically updated every `60` seconds.

- `matched`: The number of messages that are accepted by a given filter. Available for filters and similar objects (for example, a conditional rewrite rule). For example, if a filter matches a specific hostname, then the `matched` counter contains the number of messages that reached the filter from this hosts.

```
filter;demo_filter;;a;matched;28
```

- `memory_usage`: The memory used by the messages in the different queue types (in bytes). This includes every queue used by the object, including memory buffers (log-fifo) and disk-based buffers (both reliable and non-reliable). For example:

```
dst.network;d_net#0;tcp,127.0.0.1:9999;a;memory_usage;0
```

> **Note**
>
> **The memory usage (size) of queues is not equal to the memory usage (size) of the log messages in AxoSyslog. A log message can be in multiple queues, thus its size is added to multiple queue sizes. To check the size of all log messages, use `global.msg_allocated_bytes.value` metric.**

- `msg_size_max`: The current largest message size of the given source or destination.
- `msg_size_avg`: The current average message size of the given source or destination.

> **Note**
>
> When using the `msg_size_avg` and `msg_size_max` statistics, consider that message sizes are calculated as follows:
>
> - on the source side: the length of the incoming raw message
> - on the destination side: the length of the outgoing formatted message

- `not_matched` : The number of messages that are filtered out by a given filter. Available for filters and similar objects (for example, a conditional rewrite rule). For example, if a filter matches a specific hostname, then the `not_matched` counter contains the number of messages that reached the filter from other hosts, and so the filter discarded them.

> **Note**
>
> Since the `not_matched` metric applies to filters, and filters are expected to discard messages that do not match the filter condition, `not_matched` messages are not included in the `dropped` metric of other objects.
>
> ```
> filter;demo_filter;;a;not_matched;0
> ```

- `processed` : The number of messages that successfully reached their destination driver.

> **Note**
>
> **Consider that a message that has successfully reached its destination driver does not necessarily mean that the destination driver successfully delivered the messages as well. For example, a message can be written to disk or sent to a remote server after reaching the destination driver.**

- `queued` : The number of messages passed to the message queue of the destination driver, waiting to be sent to the destination.
- `stamp` : The UNIX timestamp of the last message sent to the destination.
- `suppressed` : The number of suppressed messages (if the `suppress()` feature is enabled).
- `written` : The number of messages successfully delivered to the destination. This value is calculated from other counters: `written = processed - queued - dropped` . That is, the number of messages AxoSyslog 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 AxoSyslog could not deliver the message to the destination and exceeded the number of retries).

This metric is calculated from other metrics. You cannot reset this metric directly: to reset it, you have to reset the metrics it is calculated from.

> **Note**
>
> Consider that for AxoSyslog version 3.36, the following statistics counters are only supported for the `http()` destination, or the `http()` destination and all `network()` sources and destinations, and all `file()` sources and destinations, respectively:
>
> - `msg_size_max`
> - `msg_size_avg`
> - `batch_size_max`
> - `batch_size_avg`
> - `eps_last_1h`
> - `eps_last_24h`
> - `eps_since_start`

Availability of statistics

Certain statistics are available only if the `stats(level())` global option is set to a higher value.

- Level 0 collects only statistics about the sources and destinations.
- Level 1 contains details about the different connections and log files, but has a slight memory overhead.
- Level 2 contains detailed statistics based on the hostname.
- Level 3 contains detailed statistics based on various message parameters like facility, severity, or tags.

When receiving messages with non-standard facility values (that is, higher than 23), these messages will be listed as `other` facility instead of their facility number.

Aggregated statistics

Aggregated statistics are available for different sources and destinations from different levels and upwards:

	msg_size_avg	msg_size_max	batch_size_avg	batch_size_max	eps_last_1h	eps_last_1h	eps_last_1h
network() source and destination	from level 1	from level 1	counter N/A	counter N/A	from level 1	from level 1	from level 1
file() source and destination	from level 1	from level 1	counter N/A	counter N/A	from level 1	from level 1	from level 1
http() destination	from level 0	from level 0	from level 0	from level 0	from level 0	from level 0	from level 0

17.4 Log statistics from the internal() source

> **Note**
>
> **Instead of using the statistics messages of the `internal()` source, we recommend monitoring AxoSyslog using metrics , or if it's not possible in your environment, by querying statistics .**

If the `stats(freq())` global option is higher than 0, AxoSyslog 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 AxoSyslog 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 (like source, destination, tag).

To reset the statistics to zero, use the following command: `syslog-ng-ctl stats --reset`

18 Multithreading and scaling

Configure multithreaded processing of sources and destinations, and tune AxoSyslog performance to scale across multiple CPU cores.

Starting with version 3.3, AxoSyslog 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.

18.1 Multithreading concepts

This section is a brief overview on how AxoSyslog 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 AxoSyslog uses multithreading may change in future releases. The current documentation applies to version 4.29.**

AxoSyslog 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 AxoSyslog.

The maximum number of worker threads AxoSyslog uses is the number of CPUs or cores in the host running AxoSyslog (up to 64). You can limit this value using the `--worker-threads` command-line option that sets the maximum total number of threads AxoSyslog can use, including the main AxoSyslog thread. However, the `--worker-threads` option does not affect the supervisor of AxoSyslog. 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 AxoSyslog.

When an event requiring a new thread occurs (for example, AxoSyslog receives new messages, or a destination becomes available), AxoSyslog 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()` option. 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, AxoSyslog 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 AxoSyslog, in addition to the limit set in the `--worker-threads` command-line option.

- The `logstore` destination uses separate threads for writing the messages from the journal to the logstore files, and also for timestamping. These threads are independent from the setting of the `--worker-threads` command-line option.
- Every `sql` destination uses its own thread. These threads are independent from the setting of the `--worker-threads` command-line option.

- The `java` destinations use one thread, even if there are multiple Java-based destinations configured. This thread is independent from the setting of the `--worker-threads` command-line option.

18.2 Configuring multithreading

Starting with version 3.6, AxoSyslog runs in multithreaded mode by default. You can enable multithreading in AxoSyslog 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")
     );
 };
```

18.3 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 AxoSyslog 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 AxoSyslog 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 an AxoSyslog server, you can use multiple connections to the server if you configure the AxoSyslog server to receive messages on multiple ports, and configure separate destinations on the clients to use both ports.

19 Nonsequential message processing

Process messages from a single connection on multiple threads with parallelize(), and keep message ordering where you need it with partition keys.

By default, AxoSyslog processes log messages arriving from a single connection sequentially. Sequential processing:

- ensures message ordering, and the
- efficient use CPU on a per message basis.

Sequential processing performs well if you have relatively many parallel connections, in which case it uses all the available CPU cores. However, if a small number of connections deliver a large number of messages, this behavior becomes a bottleneck.

Starting with AxoSyslog version 4.3, AxoSyslog can distribute a stream of incoming messages between a set of workers to process the stream by multiple threads in parallel. Depending on how you partition the stream, you might lose the message ordering, but can scale the incoming load to all CPUs in the system, even if the entire load is coming from a single, chatty sender.

To enable this mode of execution, use the `parallelize()` element in your log path.

The following example takes the messages of the `tcp()` source and processes them with 4 parallel threads, regardless of the number of connections used to deliver the messages to the `tcp()` source.

```
log {
  source {
    tcp(
      port(2000)
      log-iw-size(10M) max-connections(10) log-fetch-limit(100000)
    );
  };
  parallelize(workers(4));

  # from this part on, messages are processed in parallel even if
  # messages are originally coming from a single connection

  parser { ... };
  destination { ... };
};
```

`parallelize()` uses round-robin to allocate messages to workers (called partitions in versions between 4.3-4.16) by default, but you can retain ordering for a subset of messages with the `worker-partition-key()` option. The `worker-partition-key()` option specifies a template: messages that expand the template to the same value are mapped to the same partition. For example, you can partition messages based on their sender host:

```
log {
  source {
    tcp(
      port(2000)
      log-iw-size(10M) max-connections(10) log-fetch-limit(100000)
    );
  };
  parallelize(workers(4) worker-partition-key("$HOST"));

  # from this part on, messages are processed in parallel if their
  # $HOST value differs. Messages with the same $HOST will be mapped
  # to the same partition and are processed sequentially.

  parser { ... };

  destination { ... };
};
```

Staring with AxoSyslog version 4.17, you can use the `batch-size()` option to specify how many consecutive messages should be processed by a single `parallelize()` worker. This ensures that this many messages preserve their order on the destination side, and also improves `parallelize()` performance. Default value: `100` in version 4.26 and later (was disabled (`0`) in versions 4.17-4.25).

20 Writing Python modules

Write AxoSyslog destinations, parsers, template functions, and sources in Python, and add the code to DEB and RPM packages.

AxoSyslog has comprehensive support for implementing various logging components in Python. This chapter shows you how to use this functionality.

When to use Python

The Python bindings are useful if the facilities provided by the AxoSyslog configuration language is not sufficient, that is:

- AxoSyslog doesn't support a specific service (for example, API based log sources or information sources that you want to use for data enrichment purposes).
- The AxoSyslog configuration language does not support a specific transformation (in that case, please tell us about your use-case).
- You want to work on complex data structures (for example, JSON) which easier to do in a real programming language.

While Python is very powerful and you can produce clean and production ready solutions with it, the drawback is usually performance. Python code is usually slower than the native functionality that AxoSyslog provides.

To offset this impact of performance degradation, it's a good strategy to only process a subset of the incoming log stream with Python code and use native configuration elements to select which subset is traversing said Python code.

Creating and storing the Python code

You can embed Python code directly into `syslog-ng.conf` , or work with Python modules.

Embedding Python into AxoSyslog configuration

You can simply use a top-level `python {}` block to embed your Python code, like this:

```
@version: 4.0

python {

def template_function(msg):
    return b"Hello World from Python! Original message: " + msg['MSGHDR'] +
msg['MESSAGE']

};

log {
    source { tcp(port(2000)); };
    destination { file("logfile" template("$ISODATE $(python template_function)"));
};
};
```

Using Python modules

You can also put your code into a proper Python module and then use it from there. AxoSyslog automatically adds `${sysconfdir}/python` to your PYTHONPATH (normally `/etc/syslog-ng/python`), with that in mind add the following code to `/etc/syslog-ng/python/mytemplate.py`:

```
def template_function(msg):
    return b"Hello World from Python! Original message: " + msg['MSGHDR'] +
msg['MESSAGE']
```

The Python glue in AxoSyslog automatically imports modules when it encounters an identifier in dotted notation, so if you use this AxoSyslog config:

```
@version: 4.0

log {
    source { tcp(port(2000)); };
    destination { file("logfile" template("$ISODATE $(python
mytemplate.template_function)")); };
};
```

AxoSyslog recognizes that `mytemplate.template_function` is a qualified name and attempts to import `mytemplate` as a module and then looks up `template_function` within that module.

> Note: Modules are only imported once, so you will need to restart AxoSyslog for a change to take effect.

AxoSyslog reload and Python

When you reload `syslog-ng` (with `syslog-ng-ctl` reload or `systemctl reload syslog-ng`) then the `python` block in your configuration is reloaded with the rest of the configuration file. Any changes you make in Python code directly embedded in your configuration takes effect after the reload. This also means that any global variables are reset, so you cannot store state across reloads in your `python {}` block.

Modules are only imported once and kept across reloads, even if the AxoSyslog configuration is reloaded. This means that you can store global state in modules and they will be kept, even as AxoSyslog reinitializes the configuration.

In case you want to reload a module every time AxoSyslog configuration is reinitialized, you need to do this explicitly with a code similar to this:

```
python {

import mymodule
import importlib

# reload mymodule every time syslog-ng reloads
importlib.reload(mymodule)

};
```

Destination driver

You can derive a destination driver in Python from the `LogDestination` class, as defined by the `syslogng` module, like in the following example:

`mydestination.py`:

```
from syslogng import LogDestination

class MyDestination(LogDestination):
    def send(self, msg):
        return True
```

The interface of the `LogDestination` class is documented in the `syslogng.dest` module, which is stored in the `modules/python-modules/syslogng/dest.py` file of the source tree.

Once all required methods are implemented, you can use the `python` destination in the AxoSyslog configuration language.

```
destination whatever {
    python(class(mydestination.MyDestination));
};
```

There's a more complete example destination in the `python_example()` destination plugin, that is located in the directory `modules/python-modules/syslogng/modules/example/` directory within the source tree, or the same files installed under `${exec_prefix}/syslog-ng/python/syslogng/modules` in a production deployment.

Template function plugin

Template functions extend the AxoSyslog template language. They get a `LogMessage` object and return a string which gets embedded into the output of the template. You can have AxoSyslog call a Python function from the template language using the `$(python)` template function.

```
@version: 4.0

python {

def template_function(msg):
    return b"Hello World from Python! Original message: " + msg['MSGHDR'] +
msg['MESSAGE']

};

...

destination d_file {
    file("/var/log/whatever" template("$(python template_function)"));
};
```

The Python function must be callable. IT receives a `LogMessage` instance and returns a string (`str` or `bytes`). The message passed to the template function is read-only. If you are trying to change a name-value pair, you will receive an exception.

Parser plugin

You can derive parser plugins in Python from the `LogParser` class as this example shows:

```
from syslogng import LogParser

class MyParser(LogParser):

    def parse(self, msg):
        msg['name'] = 'value'
        return True
```

In contrast to template functions, parsers receive a read-writable `LogMessage` object, so you can modify its contents.

Source plugins

There are two kinds of source plugins that can be implemented in Python:

- `LogFetcher` : `LogFetcher` provides a convenient interface for fetching messages from backend services via blocking APIs, but it is limited to performing the fetching operation in a sequential manner: you fetch a batch of messages, feed them to the AxoSyslog pipeline, then repeat.
- `LogSource` : `LogSource` is more low-level but allows the use of an asynchronous framework (for example, `asyncio`) to perform message fetching along multiple threads of execution.

Both are defined by the `syslogng.source` module.

Source driver based on LogFetcher

`LogFetcher` provides a convenient interface for fetching messages from backend services via blocking APIs, but it is limited to performing the fetching operation in a sequential manner: you fetch a batch of messages, feed them to the AxoSyslog pipeline, then repeat.

For a `LogFetcher` class, you have to implement the `fetch()` method. This is the main entry point, which is automatically invoked by `syslog-ng` , whenever it consumes incoming messages.

```
@version: 4.0

python {
from syslogng import LogFetcher
from syslogng import LogMessage
import time

class MyFetcher(LogFetcher):
    def fetch(self):
        time.sleep(1)
        msg = LogMessage.parse("<5>2022-02-02T10:23:45+02:00 HOST program[pid]:
foobar", self.parse_options)
        return self.SUCCESS, msg

};


log {
    source { python-fetcher(class(MyFetcher)); };
    destination { file("messages"); };
};
```

This example generates one message every second, based on a literal string that is parsed as a syslog message.

The source is running in a dedicated thread, so it is free to block.

To limit the rate of generating messages, the `time.sleep(1)` call in the first line of the `fetch()` method sleeps for 1 second between the invocations of the method. If that sleep wasn't there, the source would produce about 100-110k messages per second, depending on the speed of your CPU, the performance of the Python interpreter and the `syslog-ng` core.

If the fetcher connects to an external API, the sleep is usually not needed, as the response time of the API is a limiting factor.

Adding persistent state

If you are fetching messages from an API, you need to keep track of which messages were already fetched. Storing the position in a variable is not a good solution, because the value of the variable is lost when `syslog-ng` is reloaded or restarted (depending on

where you store that variable, in the `python {}` block or in a module).

Use the `Persist()` class that uses the persistent state handling functionality of AxoSyslog. This allows you to persist variables in a file that gets stored in the `${localstatedir}/syslog-ng.persist` file along with the rest of the `syslog-ng` states.

```
class MyFetcher(LogFetcher):

    def init(self, options):
        self.persist = Persist("MyFetcher_persistent_data", defaults={"counter":
1})
        return True

    def fetch(self):
        time.sleep(1)
        counter = self.persist['counter']
        self.persist['counter'] += 1
        msg = LogMessage.parse("<5>2022-02-02T10:23:45+02:00 HOST program[pid]:
foobar %d" % counter, self.parse_options)
        return self.SUCCESS, msg
```

Once initialized, a `Persist()` instance behaves as a dict where you can store Python values. Currently `str`, `bytes` and `int` are supported. Anything you store in a persist instance is remembered even across restarts. The entries are backed up to disk immediately after changing them (using an `mmap()`-ped file), so you don't have to explicitly commit them to disk.

You can store position information in a `Persist()` entry, but it's not always the best choice. In AxoSyslog, producing messages is decoupled from their delivery: sometimes a message is still in-flight for a while before being delivered. This time can be significant if a destination consumes messages at a slow rate. In this case, if you store the position once fetched, the message would still be sitting in a queue waiting to be delivered. If the queue is not backed by a disk-buffer, then these messages would be lost, once `syslog-ng` is restarted.

To anticipate this case, use bookmarks .

Bookmarks in a source

The bookmarking mechanism allows messages to carry individual markers that uniquely identify a message and its position in a source stream. For example, in a source file the bookmark would contain the position of the message within that file. An API may have a

similar mechanism in place in which the source API associates an opaque to each message, which signifies its position in the repository.

A specific example for bookmarks is systemd-journald, which has a “cursor” indicating the position of each journal record. The cursor can be used to restart the reading of the log stream.

Once you’ve identified what mechanism the source offers that maps to the bookmark concept, decide how you want to track these bookmarks. Which bookmark tracking strategy you should use depends on the API specifics.

- Some APIs are sequential in nature, thus you can only acknowledge the “last fetch position” in that sequence.
- Other APIs allow you to acknowledge messages individually.

AxoSyslog supports both methods.

The following Python example updates the current position in a source stream only when the AxoSyslog destination has acknowledged the messages in the sequence (that is, when the messages were properly sent).

```
class MyFetcher(LogFetcher):
    counter = 0

    def init(self, options):
        self.persist = Persist("MyFetcher_persistent_data", defaults={"position":
0})
        self.counter = self.persist['position']

        # pass self.message_acked method as ACK callback
        self.ack_tracker = ConsecutiveAckTracker(ack_callback=self.message_acked)
        return True

    def message_acked(self, acked_message_bookmark):
        # update current persisted position when syslog-ng delivered the
        # message, but only then.
        self.persist['position'] = acked_message_bookmark

    def fetch(self):
        time.sleep(1)
        self.counter += 1

        # depending on the speed of our consumer and the setting of
        # flags(flow-control), the current counter and the acked value may
        # differ in the messages generated.
        msg = LogMessage.parse("<5>2022-02-02T10:23:45+02:00 HOST program[pid]:
foobar %d (acked so far %d)" % (self.counter, self.persist['position']),
self.parse_options)

        # this is where we set the bookmark for the message
        msg.set_bookmark(self.counter)
        return self.SUCCESS, msg
```

Acknowledgement tracking strategies

Some APIs provide simple, while others provide more complex ways to track messages that are processed. AxoSyslog provides the following strategies to cope with them.

- Instant tracking (`InstantAckTracker`): Messages are considered delivered as soon as the destination driver (or the disk-buffer) acknowledges them. Out-of-order deliveries are reported as they happen, so an earlier message may be acknowledged

later than a message originally encountered later in the source stream.

- Consecutive tracking (`ConsecutiveAckTracker`): Messages are assumed to form a stream and the bookmark is a position in that stream. Unordered deliveries are properly handled by only acknowledging messages that were delivered in order. If unordered delivery happens, the tracker waits for the sequence to fill up, that is, it waits for all preceeding messages to be delivered as well.
- Batched tracking (`BatchedAckTracker`): Messages are assumed to be independent, not forming a sequence of events. Each message is individually tracked, the source driver has the means to get delivery notifications of each and every message independently. The acknowledgements are accumulated until a timeout happens, at which point they get reported as a single batch.

You can initialize your `ack_tracker` in the `init` method, like this:

```
class MyFetcher(LogFetcher):

    ...

    def init(self, options):
        # pass self.message_acked method as ACK callback
        self.ack_tracker = ConsecutiveAckTracker(ack_callback=self.message_acked)
        return True

    def message_acked(self, acked_message_bookmark):
        pass

    def fetch(self):
        ...
        msg.set_bookmark("whatever-bookmark-value-that-denotes-position")
    ...
```

The previous example uses `ConsecutiveAckTracker` , so you get acknowledgements in the order messages were generated. The argument of the `message_acked` callback is the “bookmark” value that you set using `set_bookmark()` .

Using `InstantAckTracker` is very similar, just replace `ConsecutiveAckTracker` with `InstantAckTracker` . In this case you’d get a callback as soon as a message is delivered without preserving the original ordering.

```
class MyFetcher(LogFetcher):

    ...

    def init(self, options):
        self.ack_tracker = InstantAckTracker(ack_callback=self.message_acked)
        return True

    def message_acked(self, acked_message_bookmark):
        pass
```

While `ConsecutiveAckTracker()` seems to provide a much more useful service, `InstantAckTracker()` performs better, as it does not have to track acknowledgements of individual messages.

The most complex scenario is implemented by `BatchedAckTracker`, this allows you to track the acknowledgements for individual messages, as they happen, not enforcing any kind of ordering.

```
class MyFetcher(LogFetcher):

    ...

    def init(self, options):
        self.ack_tracker = BatchedAckTracker(timeout=500, batch_size=100,

batched_ack_callback=self.messages_acked)
        return True

    def messages_acked(self, acked_message_bookmarks):
        pass
```

`BatchedAckTracker` calls your callback periodically, as set by the `timeout` argument in milliseconds. `batch_size` specifies the number of outstanding messages at a time.

With this interface it's quite easy to send acknowledgements back to the source interface where per-message acknowledgements are needed (for example, Google PubSub).

Accessing the `flags()` option

The state of the `flags()` option is mapped to the `self.flags` variable, which is a `Dict[str, bool]`, for example:

```
{
    'parse': True,
    'check-hostname': False,
    'syslog-protocol': True,
    'assume-utf8': False,
    'validate-utf8': False,
    'sanitize-utf8': False,
    'multi-line': True,
    'store-legacy-msghdr': True,
    'store-raw-message': False,
    'expect-hostname': True,
    'guess-timezone': False,
    'header': True,
    'rfc3164-fallback': True,
}
```

Source driver based on LogSource

`LogSource` allows the use of an asynchronous framework (for example, `asyncio`) to perform message fetching along multiple threads of execution.

The following example uses `asyncio` to generate two independent sequences of messages: the first is generated every second, the other every 1.5 seconds, running concurrently via an `asyncio` event loop.

It is also easy to create a source that implements an HTTP server, and which injects messages coming via HTTP to the AxoSyslog pipeline.

```
from syslogng import LogSource
from syslogng import LogMessage
import asyncio

class MySource(LogSource):
    cancelled = False

    def run(self):
        asyncio.run(self.main())

    async def main(self):
        await asyncio.gather(self.sequence1(), self.sequence2())

    async def sequence1(self):
        while not self.cancelled:
            await asyncio.sleep(1)
            self.post_message(LogMessage("message1"))

    async def sequence2(self):
        while not self.cancelled:
            await asyncio.sleep(1.5)
            self.post_message(LogMessage("message2"))

    def request_exit(self):
        self.cancelled = True
```

Acknowledgement mechanisms (`ConsecutiveAckTracker` , `BatchedAckTracker`) and `flags()` mapping can be used similarly to how it was described at `LogFetcher` .

Making it more native config-wise

The examples so far used some form of `python()` driver, for instance, this was the Python based destination driver:

```
destination whatever {
    python(class(mydestination.MyDestination)
           options(option1 => value,
                   option2 => value));
};
```

While this works, the syntax doesn't look like other parts of the configuration, and is also hard to read. The usual syntax for referencing regular drivers is something like this:

```
destination whatever {
    my-destination(option1(value) option2(value));
};
```

To make the syntax more native, you can use the block functionality to wrap your Python driver, hide that it's actually Python, and provide a syntax to your code that is more convenient to use.

```
block destination my-destination(option1(value)
                                 option2(value)) {
    python(class(mydestination.MyDestination)
                 options(option1 => `option1`,
                         option2 => `option2`));
};
```

This block allows the use of the more AxoSyslog-native syntax, completely hiding the fact that the implementation is Python based, concentrating on functionality.

Add this wrapper to your Python module in an `scl` subdirectory as a file with a .conf extension. AxoSyslog automatically includes these files along the rest of the SCL.

Adding the code to `syslog-ng`

To add your Python-based modules to `syslog-ng`, complete the following steps.

1. Create a Python package: add the `__init__.py` file and anything that the file references to the `modules/python-modules/<name-of-your-module>` directory of the syslog-ng repository .
2. Add your files to the source tarball by listing them in the EXTRA_DIST variable of the `modules/python-modules/Makefile.am` file.
3. Run `make install` to install your module along the rest of the `syslog-ng` binaries.
4. Open a pull request.

External dependencies

If your Python code depends on third-party libraries, those need to be installed on the system where your code is deployed. If your deployment mechanism is based on DEB or RPM packages, make sure that you add these OS-level dependencies to the packages generated.

- For DEB packages, add the dependency package to the `Depends` line.
- For RPM packages, add the dependency package as a `Requires` line to the `.spec` file.

If you want to use `pip/requirements.txt` to deploy dependencies, you can invoke pip during `make install` time so that AxoSyslog's private Python directory would contain all the dependencies that you require.

Adding Python code to the DEB package

To add your module to the `syslog-ng` DEB package, complete the following steps.

1. Create a new file in `packaging/debian/` called `syslog-ng-mod-<yourmodule>.install` .
2. Populate this file with wildcard patterns that capture the files of your package after installation. For example:

```
usr/lib/syslog-ng/python/syslogng/modules/<yourmodule>/*
```

3. Add an entry to `packaging/debian/control` :

```
Package: syslog-ng-mod-<yourmodule>
Architecture: any
Multi-Arch: foreign
Depends: ${shlibs:Depends}, ${misc:Depends}, syslog-ng-core (>=
${source:Version}), syslog-ng-core (<< ${source:Version}.1~), syslog-ng-mod-
python
Description: The short description of the package
  This is a longer description with dots separating paragraphs.
  .
  This package provides a collection of example plugins.
```

4. Add your `.install` file to the tarball by adding it to the `EXTRA_DIST` in the `Makefile.am`.

Adding Python code to RPM packages

The RPM package is less modular than the Debian package and it automatically captures all Python modules in the `syslog-ng-python` package without having to list them explicitly.

If you need to customize the installation, you can find the spec file in `packaging/rhel/syslog-ng.spec` which is populated and copied to the root at tarball creation.

21 Best practices and examples

Recommendations for high message load, log rotation, name resolution, chroot environments, and load balancing between multiple destinations.

This chapter discusses some special examples and recommendations.

21.1 General recommendations

This section provides general tips and recommendations on using `syslog-ng`. Some of the recommendations are detailed in the sections below:

- Do not base the separation of log messages in different files on the `facility` parameter. As several applications and processes can use the same facility, the facility does not identify the application that sent the message. By default, the `facility` parameter is not even included in the log message itself. In general, sorting the log messages into several different files can make finding specific log messages difficult. If you must create separate log files, use the application name.
- Standard log messages include the local time of the sending host, without any time zone information. It is recommended to replace this timestamp with an ISODATE timestamp, because the ISODATE format includes the year and timezone as well. To convert all timestamps to the ISODATE format, include the following line in the `syslog-ng.conf` configuration file:

```
options {ts-format(iso) ; };
```

- Resolving the IP addresses of the clients to domain names can decrease the performance. For details, see Using name resolution in syslog-ng.

21.2 Handling large message load

This section provides tips on optimizing the performance of `syslog-ng`. Optimizing the performance is important for AxoSyslog 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.
- **Warning**

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

  As a general rule, increase the `so-rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so-rcvbuf()` at least to `2 097 152` bytes.
- Increase the value of the `flush-lines()` parameter. Increasing `flush-lines()` from `0` to `100` can increase the performance of AxoSyslog by 100%.

21.3 Using name resolution in syslog-ng

The AxoSyslog application can resolve the hostnames of the clients and include them in the log messages. However, the performance of AxoSyslog 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 AxoSyslog 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 using the `use-dns(persist_only)` option. For details, see Resolving hostnames locally.

> **Note**
>
> **Domain name resolution is important mainly in relay and server mode.**

21.3.1 Resolving hostnames locally

Purpose:

Resolving hostnames locally enables you to display hostnames in the log files for frequently used hosts, without having to rely on a DNS server. The known IP address - hostname pairs are stored locally in a file. In the log messages, AxoSyslog will replace the IP addresses of known hosts with their hostnames. To configure local name resolution, complete the following steps:

Steps:

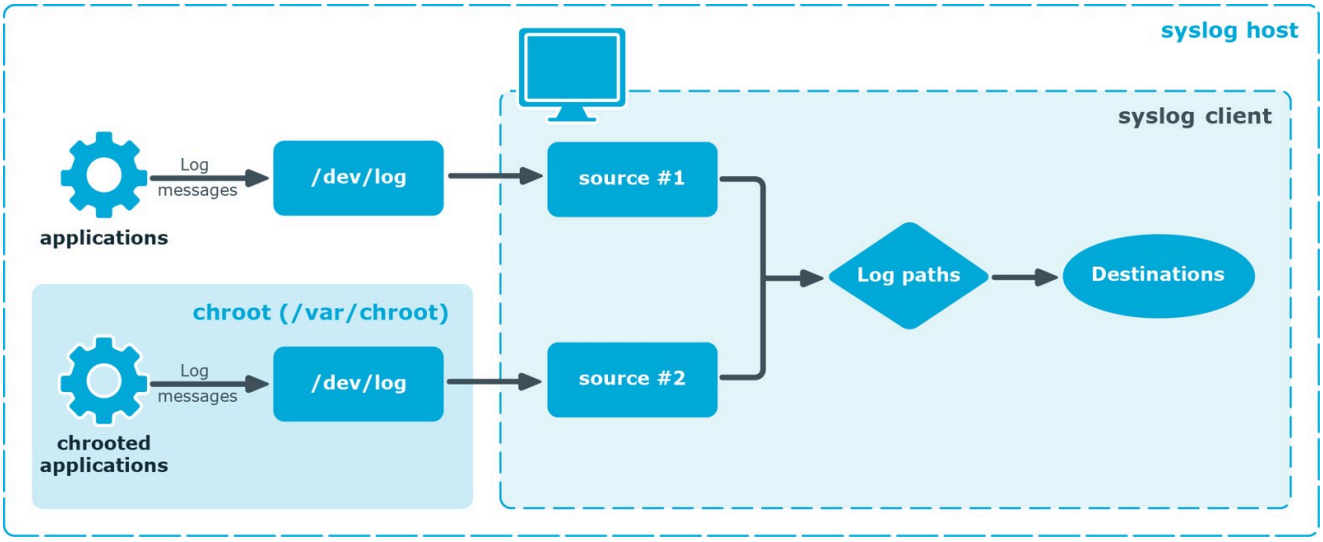
1. Add the hostnames and the respective IP addresses to the file used for local name resolution. On Linux and UNIX systems, this is the `/etc/hosts` file. Consult the documentation of your operating system for details.
2. Instruct AxoSyslog to resolve hostnames locally. Set the `use-dns()` option to `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);
};
```

21.4 Collecting logs from chroot

Purpose:

To collect logs from a chroot using an AxoSyslog client running on the host, complete the following steps:



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 AxoSyslog 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`.**

21.5 Configuring log rotation

The AxoSyslog application does not rotate logs by itself. To use AxoSyslog for log rotation, consider the following approaches:

Use logrotate together with AxoSyslog:

- It is ideal for workstations or when processing fewer logs.
- It is included in most distributions by default.
- Less scripting is required, only `logrotate` has to be configured correctly.
- Requires frequent restart (AxoSyslog must be reloaded/restarted when the files are rotated). After rotating the log files, reload AxoSyslog using the `syslog-ng-ctl reload` command, or use another method to send a SIGHUP to AxoSyslog.
- The statistics collected by AxoSyslog, and the correlation information gathered with Pattern Database, are lost with each restart.

Separate incoming logs based on time, host or other information:

- It is ideal for central log servers, where regular restart of AxoSyslog 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)
     );
 };
```

Example: Logstore destination for log rotation

This sample logstore destination configuration stores incoming logs in logstores that are named based on the current year, month and day, and places these logstores in directories that are named based on the hostname:

```
destination d_logstore {
     logstore(
         "/var/log/remote/${HOST}/${YEAR}_${MONTH}_${DAY}.lgs"
         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 {} \;
```

21.6 Load balancing logs between multiple destinations

These sections describe a method of load balancing logs between multiple AxoSyslog destinations. The first subsection describes the round robin load balancing method based on the `R_MSEC` macro of AxoSyslog, while the second subsection describes a configuration generator that you can use as an alternative to using the example configuration described in the first subsection.

For more information about the `R_MSEC` macro and further macros of AxoSyslog, see Macros of AxoSyslog.

21.6.1 Load balancing with a round robin load balancing method based on the R_MSEC macro

This section describes a round robin load balancing method based on the `R_MSEC` macro of AxoSyslog to load balance your logs between multiple AxoSyslog destinations.

> **Note**
>
> **If `R_MSEC` is not precise enough, you can replace it with `R_USEC` (which uses microseconds instead of milliseconds).**

For more information about the `R_MSEC` macro and further macros of AxoSyslog, see Macros of AxoSyslog.

Example: round robin load balancing between multiple destinations

The following example is a round-robin load balancing method, based on AxoSyslog's `R_MSEC` macro.

```
   destination d_lb_network {
      channel {
        channel {
          filter {
          "0" == "$(% ${R_MSEC} 2)"
          };
          destination {
            network("myhost1"
              disk-buffer(flow-control-window-size(10000) capacity-
bytes(2000000)));
          };
          flags(final);
        };

        channel {
        filter {
        "1" == "$(% ${R_MSEC} 2)"
        };

        destination {
          network("myhost2"
            disk-buffer(flow-control-window-size(10000) capacity-bytes(2000000)));
        };
        flags(final);
        };
      };
    };
```

The `filter {" <return value >" == "$(% ${R_MSEC} 2)"};` code snippets (in bold) serve as the basis of the method. This filter separates incoming log messages' timestamp values based on the `R_MSEC` macro, using a division with remainder method, and distributes the log messages equally between two destinations based on the return value (in this case, `0` or `1`).

If you need a file instead of a network destination, replace the network destination with the file in the example (and use the same analogy for any other AxoSyslog destinations).

For an alternative method to use the round robin load balancing method based on the `R_MSEC` macro, see Configuration generator for the load balancing method based on MSEC hashing.

21.6.2 Configuration generator for the load balancing method based on MSEC hashing

This section describes a configuration generator SCL for the load balancing method based on MSEC hashing to load balance your logs between multiple AxoSyslog destinations.

Warning

Consider that `network-load-balancer()` is not a destination, only a script that generates the example configuration described in Load balancing with a round robin load balancing method based on the R_MSEC macro.

Also consider that the configuration generator script may change incompatibly in the future. As a result, Axoflow does not officially support using this script, and recommends that you only use this script at your own risk.

As an alternative to using the example configuration described in Load balancing with a round robin load balancing method based on the R_MSEC macro, a configuration generator script is also available in AxoSyslog:

```
destination d_lb {network-load-balancer(targets(myhost1 myhost2 myhost3))};
```

Where destinations share the same configuration except for the destination address, balancing is based on MSEC hashing.

Starting with version 4.24, `network-load-balancer()` supports the `failover()` option .

22 Troubleshooting

Diagnose lost messages, error messages, and crashes, collect debugging data and core files, and recover data from orphaned diskbuffer files.

This chapter provides tips and guidelines about troubleshooting problems related to `syslog-ng`.

- As a general rule, first try to log the messages to a local file. Once this is working, you know that AxoSyslog 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 AxoSyslog server does not receive the messages, verify that the IP addresses and ports are correct in your sources and destinations. Also, check that the client and the server uses the same protocol (a common error is to send logs on UDP, but configure the server to receive logs on TCP).

  If the problem persists, use `tcpdump` or a similar packet sniffer tool on the client to verify that the messages are sent correctly, and on the server to verify that it receives the messages.
- To find message-routing problems, run AxoSyslog with the following command `syslog-ng -Fevd`. That way AxoSyslog will run in the foreground, and display debug messages about the messages that are processed.
- If AxoSyslog is closing the connections for no apparent reason, be sure to check the log messages of `syslog-ng`. You may also want to run `syslog-ng` with the `--verbose` or `--debug` command-line options for more-detailed log messages. You can enable these messages without restarting `syslog-ng` using the `syslog-ng-ctl verbose --set=on` command. For details, see the The syslog-ng.conf 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 AxoSyslog client and the source of your AxoSyslog server, everything should work as expected. Unfortunately, there are some other combinations, that may seem to work, but result in losing parts of the messages. For details on the working combinations, see Things to consider when forwarding messages between AxoSyslog hosts.
- If you're using FilterX, see Troubleshooting FilterX for specific tips.

Support

In case you need help with any of the AxoSyslog projects, or directly with `syslog-ng`, you have several ways to contact us:

- Open a GitHub issue in the AxoSyslog repository .
- Chat with us in the syslog-ng or axosyslog channels of the Axoflow Discord server .
- Fill the Axoflow contact form .

We also provide consulting and professional services for logging and observability related projects. Contact us if you need our help!

22.1 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 AxoSyslog does its best to avoid message loss. Usually losing messages can be avoided with careful planning and proper configuration of AxoSyslog 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 AxoSyslog client: Make sure to use an appropriate source to receive the logs from the application (for example, from `/dev/log`).
- When AxoSyslog is sending messages: If AxoSyslog cannot send messages to the destination and the output buffer gets full, AxoSyslog will drop messages.

  Use flags (flow-control) to avoid this (for details, see Configuring flow-control). For more information about the error caused by the missing flow-control, see `Destination queue full` in Error messages.

  The number of dropped messages is displayed per destination in the log message statistics of AxoSyslog (for details, see Statistics and metrics of AxoSyslog).
- On the network: When transferring messages using the UDP protocol, messages may be lost without any notice or feedback – such is the nature of the UDP protocol. Always use the TCP protocol to transfer messages over the network whenever possible.

  For details on minimizing message loss when using UDP, see our blog posts about soslog over UDP .
- 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 AxoSyslog 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 AxoSyslog is receiving messages: The receiving AxoSyslog (for example, the AxoSyslog 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 AxoSyslog (for details, see Statistics and metrics of AxoSyslog).

- When the destination cannot handle large load: When AxoSyslog 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 AxoSyslog fill up, AxoSyslog 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 AxoSyslog server: If the host running the AxoSyslog server experiences an unclean shutdown, it takes time until the clients realize that the connection to the AxoSyslog 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 AxoSyslog is writing messages into files: If AxoSyslog 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.

22.2 Creating 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.

22.3 Collecting debugging information with strace, truss, or tusc

To properly troubleshoot certain situations, it can be useful to trace which system calls AxoSyslog performs. How this is performed depends on the platform running AxoSyslog. In general, note the following points:

- When AxoSyslog 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`

> **Note**
>
> **To execute these commands on an already running AxoSyslog process, use the `-p <pid_of_syslog-ng>` parameter.**

22.4 Running a failure script

Purpose:

You can create a failure script that is executed when AxoSyslog terminates abnormally, that is, when it exits with a non-zero exit code. For example, you can use this script to send an automatic email notification.

Prerequisites:

The failure script must be the following file: `/opt/syslog-ng/sbin/syslog-ng-failure`, and must be executable.

To create a sample failure script, 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 AxoSyslog 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 AxoSyslog 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.

22.5 Stopping the syslog-ng process

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` .

22.6 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 AxoSyslog version. For details, see the The syslog-debun manual page. For support contacts, see Getting support.

22.7 Recover data from orphaned diskbuffer files

When you change the configuration of a AxoSyslog host that uses disk-based buffering (also called disk queue), AxoSyslog may start new disk buffer files for the destinations that you have changed. In this case, AxoSyslog 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.

22.8 Error messages

This section describes the most common error messages.

Destination queue full

Error message:	`Destination queue full, dropping messages; queue_len='10000', log_fifo_size='10000', count='4', persist_name='afsocket_dd_qfile(stream,serverdown:514)'`
Description:	This message indicates message loss. Flow-control must be enabled in the log path. When flow-control is enabled, `syslog-ng` will stop reading messages from the sources of the log statement if the destinations are not able to process the messages at the required speed. If flow-control is enabled, `syslog-ng` will only drop messages if the destination queues/window sizes are improperly sized.
Solution:	Enable flow-control in the log path. If flow-control is disabled, `syslog-ng` will drop messages if the destination queues are full. Note that `syslog-ng` will drop messages even if the server is alive. If the remote server accepts logs at a slower rate than the sender `syslog-ng` receives them, the sender `syslog-ng` will fill up the destination queue, then drop the newer messages. Sometimes this error occurs only at a specific time interval, for example, only between `7:00` AM and `8:00` AM or between `16:00` PM and `17:00` PM when your users log in or log off and that generates a lot of messages within a short interval. For more information, see Managing incoming and outgoing messages with flow-control.

Alert unknown CA

Error message:	`SSL error while writing stream; tls_error='SSL routines:ssl3_read_bytes:tlsv1 alert unknown ca'`
Description:	This message indicates that the other (remote) side could not verify the certificate sent by `syslog-ng` .
Solution:	Check the logs on the remote site and identify why the receiving `syslog-ng` could not find the CA certificate that signed this certificate.

PEM routines:PEM_read_bio:no start line

Error message:	`testuser@thor-x1:~/cert_no_start_line/certs$ openssl x509 -in cert.pem -text unable to load 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 `^M` characters as shown in the image below: 

Solution:

- On Windows, save the certificate using UTF-8, for example, using Notepad++.

  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.

22.9 Error: unexpected LL_IDENTIFIER

A missing module or SCL package makes AxoSyslog fail to start with a syntax error. Find out which package provides the driver you configured.

AxoSyslog fails to start, and reports a syntax error that points at the name of a source, destination, parser, or other configuration object:

```
syslog-ng[19147]: Error parsing config, syntax error, unexpected LL_IDENTIFIER,
expecting '}' in /etc/syslog-ng/conf.d/auditd.conf:1:19-1:30:
syslog-ng[19147]: 1-----> source s_auditd { linux-audit(); };
syslog-ng[19147]: 1----->                   ^^^^^^^^^^^
```

`LL_IDENTIFIER` means that the parser found a name it doesn't recognize in that position. There are three common causes.

The driver name is misspelled

Check the name against the reference documentation. Some drivers have names that are easy to guess incorrectly, for example, the source that reads the systemd journal is `systemd-journal()`, not `systemd-journald()`.

The module that provides the driver isn't installed

AxoSyslog is modular: most drivers live in separate packages that you install only if you need them. If the module isn't installed, its driver name is unknown to the configuration parser, and you get the same error you'd get for a typo.

Check the Install AxoSyslog on Debian/Ubuntu or the Install AxoSyslog on RHEL/Fedora/AlmaLinux page for the complete list of modules and the package that provides each of them. The Prerequisites section of every driver's page also names the package you need.

To list the modules that are currently loaded, run:

```
syslog-ng --version
```

The output contains an `Available-Modules:` line.

The SCL files aren't installed

Many drivers, like `linux-audit()`, `elasticsearch-http()`, or `telegram()`, aren't compiled modules but configuration snippets from the AxoSyslog Configuration Library (SCL).

- On Debian and Ubuntu, the SCL files are in the `axosyslog-scl` package. The `axosyslog` metapackage depends on it, but if you installed only `axosyslog-core`, you have to install it separately:

  ```
  sudo apt install axosyslog-scl
  ```

- On RHEL and compatible distributions, the SCL files are part of the `axosyslog` base package.

Your configuration must also include the SCL:

```
@version: 4.29
@include "scl.conf"
```

The default `/etc/syslog-ng/syslog-ng.conf` shipped with the packages already contains this line. You don't need to include individual SCL files: `scl.conf` loads every installed SCL plugin, including the Python-based ones from the `axosyslog-mod-python` (Debian/Ubuntu) or `axosyslog-python` (RHEL) package.

> **Note**
>
> **If you replaced the default configuration file, make sure `@include "scl.conf"` is still present. Without it, none of the SCL-based drivers are available, even if the `axosyslog-scl` package is installed.**

22.10 SELinux prevents using the execmem access on a process

If you are using a recent enough PCRE library, AxoSyslog will automatically use the JIT of the regexp engine, which will result in a similar error:

```
   setroubleshoot [21631 ] : SELinux is preventing <syslog-ng path> from using the
execmem access on a process. (...)

    python [21631 ] : SELinux is preventing <syslog-ng path> from using the execmem
access on a process.
```

To resolve this issue, switch off the PCRE JIT compile function by using the disable-jit `flags()` option in the given filter or rewrite rule of your configuration.

23 Getting support

Ways to get help with AxoSyslog and syslog-ng: GitHub issues, the Axoflow Discord server, and Axoflow professional services.

In case you need help with any of the AxoSyslog projects, or directly with `syslog-ng`, you have several ways to contact us:

- Open a GitHub issue in the AxoSyslog repository .
- Chat with us in the syslog-ng or axosyslog channels of the Axoflow Discord server .
- Fill the Axoflow contact form .

We also provide consulting and professional services for logging and observability related projects. Contact us if you need our help!

24 Manual pages

Command-line reference for syslog-ng, syslog-ng-ctl, syslog-ng.conf, loggen, pdbtool, dqtool, persist-tool, and the secure logging tools.

24.1 The syslog-ng manual page

Name

`syslog-ng` – syslog-ng system logger application

Synopsis

```
syslog-ng [options]
```

Description

The AxoSyslog application is a flexible and highly scalable system logging application. Typically, AxoSyslog 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 clients - all run AxoSyslog, and collect the log messages from the various applications, files, and other sources. The clients send all important log messages to the remote AxoSyslog server, where the server sorts and stores them.

Options

- `--caps`

  Run AxoSyslog process with the specified POSIX capability flags.

  - If the `--no-caps` option is not set, AxoSyslog has been compiled with the `--enable-linux-caps compile` option, and the host supports `CAP_SYSLOG`, AxoSyslog 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`, AxoSyslog 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.

- `--check-startup`: Use this option to perform a complete configuration initialization with `syslog-ng`, then exit with exit code indicating the result. You can use this option to check if the configuration is semantically valid and that `syslog-ng` can actually start. (The `--syntax-only` option catches only syntactical errors.)

  You can also use `syslog-ng --check-startup` in a Kubernetes environment, to run it as a dedicated configuration check container.

  Available only in AxoSyslog version 4.5 and later.

  Using this option initializes things like network listeners, so it will probably *not* work if another `syslog-ng` instance running in the background (because in that case the network address is already in use).

- `--control <file>` or `-c <file>`

  Set the location of the `syslog-ng` control socket. Default value: `/var/run/syslog-ng.ctl`

- `--config-id`

  Parse the configuration file, print the configuration identifier, then exit. For details, see Configuration identifier.

- `--debug` or `-d`

  Start `syslog-ng` in debug mode.

- `--enable-core`

  Enable AxoSyslog 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, AxoSyslog starts from the current directory (`$CWD`) so it can create core files (normally, AxoSyslog starts from `/$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.

- `--interactive` or `-i`

  Enable interactive mode.

- `--list-paths`

  Parse the configuration file, print the files and directories that it refers to, then exit without starting AxoSyslog. Available in AxoSyslog 4.29 and later.

- `--log-level <level>` or `-L <level>`

  Set the internal log level of AxoSyslog to `default` , `verbose` , `debug` , or `trace` . Available in AxoSyslog 4.0 and later.

- `--metrics-registry`

  Available in AxoSyslog 4.24 and later.

  Shows the list of available metrics. For a list of supported metrics, see Metrics reference.

- `--module-path <path>`

  The colon-separated list of directories to search for modules.

- `--module-registry`

  Display the list and description of the available modules. Note that not all of these modules are loaded automatically.

- `--no-caps`

  Run AxoSyslog as root, without capability-support. This is the default behavior. On Linux, it is possible to run AxoSyslog as non-root with capability-support if AxoSyslog was compiled with the `--enable-linux-caps` option enabled. (Run `syslog-ng --version` to display the list of enabled build parameters.)

  To run AxoSyslog with specific capabilities, use the `--caps` option.

- `--no-module-discovery`

  Disable automatic module discovery. When you use this option, the configuration file must load every module explicitly with the `@module <modulename>` statement.

- `--perf-profiling`

  Enable Linux `perf` based profiling.

- `--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 AxoSyslog 4 F1 and later.

- `--process-mode <mode>`

  Sets how to run AxoSyslog: 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 AxoSyslog if it crashes.

- `--startup-debug` or `-r`

  Enable debug logging during startup.

- `--stderr` or `-e`

  Log internal messages of AxoSyslog to `stderr`. Mainly used for debugging purposes in conjunction with the `--foreground` option. If not specified, `syslog-ng` logs such messages to its internal source.

- `--syntax-only` or `-s`

  Verify that the configuration file is syntactically correct and exit.

- `--trace` or `-t`

  Enable trace messages. Trace messages are needed mostly for finding software errors.

- `--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 AxoSyslog.

- `--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.

- `--worker-threads`

  Sets the number of worker threads AxoSyslog can use, including the main AxoSyslog thread. Note that certain operations in AxoSyslog can use threads that are not limited by this option. This setting has effect only when AxoSyslog is running in multithreaded mode. Available only in AxoSyslog 4 F1 and later. See `The AxoSyslog 7 Administrator Guide` for details.

Files

`/opt/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng.conf`

See also

syslog-ng.conf.5

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.2 The syslog-ng.conf manual page

Name

`syslog-ng.conf` – configuration file

Synopsis

`syslog-ng.conf`

Description

The AxoSyslog application is a flexible and highly scalable system logging application. Typically, AxoSyslog 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 AxoSyslog, and collect the log messages from the various applications, files, and other sources. The clients send all important log messages to the remote AxoSyslog server, where the server sorts and stores them.

Basic concepts of AxoSyslog

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

Log messages enter AxoSyslog in one of the defined sources, and are sent to one or more destinations.

Sources and destinations are independent objects, log paths define what AxoSyslog 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 AxoSyslog 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, AxoSyslog 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 AxoSyslog

- 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.conf` 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 AxoSyslog configuration file are case sensitive.

    > **Note**
    >
    > **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.**

    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 AxoSyslog should use DNS resolution to resolve IP addresses.

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

    ```
    #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 AxoSyslog 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 AxoSyslog 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 AxoSyslog configuration file:

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

The sources, destinations, and filters available in AxoSyslog are listed below. For details, see the AxoSyslog documentation .

Table: Available source drivers

Name	Description
lidarr(), prowlarr(), radarr(), readarr(), sonarr(), whisparr()	Collect logs of *arr media management applications
axosyslog-otlp()	Receive logs from another node using OpenTelemetry
darwin-oslog(), darwin-oslog-stream()	Collect native macOS system logs
default-network-drivers()	Receive and parse common syslog messages
ehttp()	Receive logs over HTTP or HTTPS

elasticsearch-bulk()	Receive messages from clients of the Elasticsearch Bulk API
file()	Collect messages from text files
hypr-audit-trail(), hypr-app-audit-trail()	Fetch events from the Hypr REST API
internal()	Collect internal messages
jellyfin()	Collect Jellyfin logs
kubernetes()	Collect and parse messages in the Kubernetes CRI (Container Runtime Interface) format
linux-audit()	Collect messages from Linux audit logs
mbox()	Convert local email messages to log messages
mqtt()	Fetch messages from MQTT brokers
network()	Collect messages using the RFC3164 protocol
nodejs()	Receive JSON messages from nodejs applications
openbsd()	Collect kernel log messages on OpenBSD systems
opentelemetry()	Receive logs, metrics, and traces from OpenTelemetry clients over the OpenTelemetry Protocol (OTLP/gRPC)
osquery()	Collect and parse osquery result logs
pacct()	Collect process accounting logs on Linux
pihole-ftl()	Collect Pi-hole FTL logs
pipe()	Collect messages from named pipes
program()	Receive messages from external applications
python()	Server-style Python source that receives messages
python-fetcher()	Write a fetcher-style Python source
qbittorrent()	Collect qBittorrent logs
snmptrap()	Read Net-SNMP traps
splunk-hec()	Receive messages sent to the Splunk HTTP Event Collector (HEC)
stdin()	Collect messages from the standard input stream
sun-streams()	Collect messages on Sun Solaris

syslog()	Collect messages using the IETF-syslog protocol
system()	Collect the system-specific log messages of a platform
systemd-journal()	Collect messages from the systemd-journal system log storage
systemd-syslog()	Collect systemd messages using a socket
tcp(), tcp6(), udp(), udp6()	OBSOLETE - Collect messages from remote hosts using the BSD syslog protocol
unix-stream(), unix-dgram()	Collect messages from UNIX domain sockets
webhook(), webhook-json()	Receive logs via a HTTP webhook
wildcard-file()	Collect messages from multiple text files

Table 2. Available destination drivers

Name	Description
amqp()	Publish messages using AMQP
arrow-flight()	Send messages to an Apache Arrow Flight server
azure-monitor()	Send messages to Azure Monitor and Sentinel
bigquery()	Send messages to Google BigQuery
clickhouse()	Send messages to a ClickHouse database
collectd()	Send metrics to collectd
discord()	Send alerts and notifications to Discord
elasticsearch2()	DEPRECATED - Send messages directly to Elasticsearch version 2.0 or higher
elasticsearch-datastream()	Send messages to Elasticsearch data streams
elasticsearch-http()	Send messages to Elasticsearch using the HTTP Bulk API
file()	Store messages in plain-text files
google-pubsub-grpc()	Send messages to Google Pub/Sub via gRPC

google-pubsub()	Send messages to Google Pub/Sub via HTTP
graphite()	Send metrics to Graphite
graylog2()	Send logs to Graylog
hdfs()	Store messages on the Hadoop Distributed File System (HDFS)
java()	Post messages over HTTP using Java
http()	Post messages over HTTP without Java
kafka()	Publish messages to Apache Kafka
kafka-c()	Publish messages to Apache Kafka (C implementation)
loggly()	Send logs to Loggly
logmatic()	Send logs to Logmatic.io
logscale()	Send messages to Falcon LogScale
loki()	Send messages to Grafana Loki
mongodb()	Store messages in a MongoDB database
mqtt()	Send messages to an MQTT broker
network()	Send messages to a remote log server using the RFC3164 protocol
openobserve-log()	Send messages to OpenObserve
opensearch()	Send messages to OpenSearch
osquery()	Send log messages to osquery's syslog table
opentelemetry()	Send logs, metrics, and traces to OpenTelemetry
pipe()	Send messages to named pipes
program()	Send messages to external applications
pseudofile()	Deliver messages to special files
python()	Write custom Python destinations
redis()	Store name-value pairs in Redis
riemann()	Monitor your data with Riemann
s3()	Send log messages to Amazon Simple Storage Service (S3)
slack()	Send alerts and notifications to a Slack channel

smtp()	Generate SMTP messages (emails) from logs
snmp()	Send SNMP traps
splunk-hec-event(), splunk-hec-raw()	Send messages to Splunk HEC
sql()	Store messages in an SQL database
stdout()	Send messages to standard output
stomp()	Publish messages using STOMP
sumologic-http(), sumologic-syslog()	Send messages to Sumo Logic
syslog()	Send messages to a remote logserver using the IETF-syslog protocol
syslog-ng()	Forward logs to another syslog-ng node
axosyslog-otlp()	Forward logs to another node using OpenTelemetry
tcp(), tcp6(), udp(), udp6()	OBSOLETE - Send messages to a remote log server using the legacy BSD-syslog protocol
telegram()	Send messages to Telegram
unix-stream(), unix-dgram()	Send messages to UNIX domain sockets
usertty()	Send messages to a user terminal

Table 3. Filter functions

Name	Description
facility()	Filter messages based on the sending facility.
filter()	Call another filter function.
host()	Filter messages based on the sending host.
in-list()	File-based whitelisting and blacklisting.
level() or priority()	Filter messages based on their priority.
match()	Use a regular expression to filter messages based on a specified header or content field.

message()	Use a regular expression to filter messages based on their content.
netmask()	Filter messages based on the IPv4 address of the sending host.
netmask6()	Filter messages based on the IPv6 address of the sending host.
program()	Filter messages based on the sending application.
rate-limit()	Limit messages rate based on arbitrary keys in each message.
source()	Select messages of the specified AxoSyslog 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

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.3 The syslog-ng-ctl manual page

Manual page of syslog-ng-ctl, which displays message statistics, reloads the configuration, and turns on debug and trace messages.

New to AxoSyslog? AxoSyslog is a binary compatible `syslog-ng` replacement, from the original creator, developed by the same team.

Same architecture, same config files, same paths. Cloud-native images, fast releases, modern observability (OTel, K8s), and powerful data processing: read more about the differences between AxoSyslog and `syslog-ng` .

Also, it's super easy to install , and you can upgrade from `syslog-ng` in minutes .

Name

`syslog-ng-ctl` – Display message statistics and enable verbose, debug and trace modes

Synopsis

```
syslog-ng-ctl [command] [options]
```

Description

> **Note**
>
> **The `syslog-ng-ctl` application is distributed with the AxoSyslog system logging application, and is usually part of the AxoSyslog package.**

The `syslog-ng-ctl` application is a utility that can:

- enable/disable various AxoSyslog messages for troubleshooting
- display statistics about the processed messages
- handling password-protected private keys
- display the currently running configuration of AxoSyslog
- reload the configuration of AxoSyslog.

Enabling troubleshooting messages

`syslog-ng-ctl log-level <level>`

Available in AxoSyslog 4.0 and later.

Use the `syslog-ng-ctl log-level <level>` command to display verbose, trace, or debug messages. If you are trying to solve configuration problems, the verbose (and occasionally debug) messages are usually sufficient. Trace messages are needed mostly for finding software errors. After solving the problem, do not forget to return the log level to the default using the `syslog-ng-ctl log-level default` command.

Use `syslog-ng-ctl log-level` without any parameters to display the current log level.

If AxoSyslog was started with the `--stderr` or `-e` option, the messages will be sent to `stderr`. If not specified, AxoSyslog will log such messages to its internal source.

If you need to use a non-standard control socket to access `syslog-ng`, use the `syslog-ng-ctl <command> --control=<socket>` command to specify the socket to use.

Higher log-levels automatically include messages from lower log-levels:

- `default`: Just normal log messages.
- `verbose`: Normal and verbose log messages.
- `debug`: Include debug messages of AxoSyslog.
- `trace`: Include trace messages of how messages are processed.

Example

```
syslog-ng-ctl log-level verbose
```

To temporarily change the log levels and access the logs of `syslog-ng`, see also the `attach` command.

Enable a single message level

`syslog-ng-ctl verbose [options]`

`syslog-ng-ctl debug [options]`

`syslog-ng-ctl trace [options]`

The `verbose`, `debug`, and `trace` commands enable or query a single message level individually. Use the `log-level` command instead when possible, because it sets the log level of AxoSyslog as a whole.

Use the command without any options to query the current state of the given message level.

These commands have the following options:

- `--control=<socket>` or `-c`

  Specify the socket to use to access `syslog-ng`. Only needed when using a non-standard socket.

- `--set=<on|off|0|1>` or `-s`

  Enable or disable the messages of the given level.

For example, the following command enables debug messages:

```
syslog-ng-ctl debug --set=on
```

Monitor AxoSyslog metrics

AxoSyslog provides detailed metrics about its performance and status for observability and monitoring. We recommend using Prometheus to scrape these metrics, see Collect metrics with Prometheus for details. To display the current metrics locally in Prometheus-compatible format, run:

```
syslog-ng-ctl stats prometheus
```

Note that which metrics are shown depends on the current value of the `stats(level())` global option (you can list the available metrics by running `syslog-ng --metrics-registry`). For details on what the metrics mean, see Metrics reference.

Example output on `stats(level(0))`:

```
syslogng_last_config_file_modification_timestamp_seconds 1764166030
syslogng_last_successful_config_reload_timestamp_seconds 1775651576
syslogng_memory_queue_events{address="192.168.0.111:514",driver="tcp",id="d_net#0",
transport="tcp"} 0
syslogng_memory_queue_processed_events_total{address="192.168.0.111:514",driver="tc
p",id="d_net#0",transport="tcp"} 19
syslogng_internal_events_total{result="dropped"} 0
syslogng_internal_events_total{result="processed"} 2
syslogng_internal_events_total{result="queued"} 0
syslogng_output_unreachable{id="d_net#0",driver="tcp",transport="tcp",address="192.
168.0.111:514"} 0
syslogng_memory_queue_memory_usage_bytes{address="192.168.0.111:514",driver="tcp",i
d="d_net#0",transport="tcp"} 0
syslogng_stats_level 0
syslogng_output_events_total{address="192.168.0.111:514",driver="tcp",id="d_net#0",
transport="tcp",result="dropped"} 0
syslogng_output_events_total{address="192.168.0.111:514",driver="tcp",id="d_net#0",
transport="tcp",result="queued"} 0
syslogng_output_events_total{address="192.168.0.111:514",driver="tcp",id="d_net#0",
transport="tcp",result="delivered"} 19
syslogng_last_config_reload_timestamp_seconds 1775651576
syslogng_memory_queue_capacity{address="192.168.0.111:514",driver="tcp",id="d_net#0
",transport="tcp"} 1000
syslogng_input_events_total{driver="journal",id="s_src#0"} 17
syslogng_scratch_buffers_bytes 0
syslogng_input_window_full_total{driver="journal",id="s_src#0"} 0
syslogng_input_window_full_total{id="s_src#1"} 0
syslogng_input_event_bytes_total{id="s_src#1"} 0
syslogng_input_event_bytes_total{driver="journal",id="s_src#0"} 0
syslogng_internal_events_queue_capacity 10000
syslogng_input_events_total{id="s_src#1"} 2
syslogng_scratch_buffers_count 14
```

The `stats prometheus` 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` and the `memory_usage` counters. (The `queued` counters show the number of messages in the message queue of the destination driver, waiting to be sent to the destination. The `memory_usage` counters show the amount of 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))

- `--remove-orphans`

  Safely removes all counters that are not referenced by any `syslog-ng stat` producer objects.

  The flag can be used to prune dynamic and static counters manually. This is useful, for example, when a templated file destination produces a lot of stats. We recommend using `syslog-ng-ctl stats --remove-orphans` during each configuration reload, but only after the values of those metrics have been scraped by all scrapers.

```
dst.file;#anon-destination0#0;/tmp/2021-08-16.log;o;processed;253592
dst.file;#anon-destination0#0;/tmp/2021-08-17.log;o;processed;156
dst.file;#anon-destination0#0;/tmp/2021-08-18.log;a;processed;961
```

- `--with-legacy-metrics`

  Display legacy metrics in Prometheus-compatible format.

> **Note**
>
> **The `stats(lifetime())` global option can be used to do the same automatically and periodically, but `stats(lifetime())` removes only dynamic counters that have a timestamp field set.**

Access legacy statistics

The `syslog-ng-ctl stats` and the `syslog-ng-ctl query` commands give you access to the legacy statistics of AxoSyslog. There are several newer metrics that aren't available in these statistics, so we recommend using metrics instead.

- `syslog-ng-ctl query` gives structured access to the selected legacy statistics.
- `syslog-ng-ctl stats` lists all the available legacy statistics in bulk.

Exactly which statistics are available depends on your AxoSyslog configuration (that is, the sources, destinations, and other objects you have configured), and also on your `stats(level())` global option settings.

Query legacy statistics

The AxoSyslog application stores various 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 AxoSyslog configuration (that is, the sources, destinations, and other objects you have configured), and also on your `stats(level())` global option settings.

List legacy statistics

`syslog-ng-ctl query list`

Use the `syslog-ng-ctl query list` command to display the list of metrics that AxoSyslog collects about the processed messages.

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, except for the `queued` and the `memory_usage` counters. (The `queued` counters show the number of messages in the message queue of the destination driver, waiting to be sent to the destination. The `memory_usage` counters show the amount of 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))

Get selected 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 statistics 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 AxoSyslog instance.

- `--reset`

  Use `--reset` to set the selected counters to 0 after executing the query, except for the `queued` and the `memory_usage` counters. (The `queued` counters show the number of messages in the message queue of the destination driver, waiting to be sent to the destination. The `memory_usage` counters show the amount of 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))

The stats command

`syslog-ng-ctl stats [options]`

Use the `stats` command to display the legacy statistics about the processed messages. Exactly which statistics are available depends on your AxoSyslog configuration (that is, the sources, destinations, and other objects you have configured), and also on your `stats(level())` global option settings.

Note that starting with version 4.14, AxoSyslog automatically shows orphan counters to avoid losing information. Information loss could happen, for example, when sending messages using short-lived (few seconds long) connections, while scraping metrics in minute intervals.

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 `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` and the `memory_usage` counters. (The `queued` counters show the number of messages in the message queue of the destination driver, waiting to be sent to the destination. The `memory_usage` counters

show the amount of 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))

- `--remove-orphans`

  Safely removes all counters that are not referenced by any `syslog-ng stat` producer objects.

  The flag can be used to prune dynamic and static counters manually. This is useful, for example, when a templated file destination produces a lot of stats. We recommend using `syslog-ng-ctl stats --remove-orphans` during each configuration reload, but only after the values of those metrics have been scraped by all scrapers.

```
dst.file;#anon-destination0#0;/tmp/2021-08-16.log;o;processed;253592
dst.file;#anon-destination0#0;/tmp/2021-08-17.log;o;processed;156
dst.file;#anon-destination0#0;/tmp/2021-08-18.log;a;processed;961
```

- `--with-legacy-metrics`

  Display legacy metrics in Prometheus-compatible format.

> **Note**
>
> **The `stats(lifetime())` global option can be used to do the same automatically and periodically, but `stats(lifetime())` removes only dynamic counters that have a timestamp field set.**

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 AxoSyslog 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.

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 AxoSyslog 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 AxoSyslog cannot use it. The sources and destinations that use these keys will not work until you provide the passwords. Other parts of the AxoSyslog configuration will be unaffected. You must provide the passphrase of the password-protected keys every time AxoSyslog 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. AxoSyslog 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 AxoSyslog configuration file.

- `--secret=<passphrase-of-the-key>` or `-s`

  The password or passphrase of the private key.

Displaying the configuration

`syslog-ng-ctl config [options]`

Use the `syslog-ng-ctl config` command to display the configuration that AxoSyslog is currently running. By default, only the content of the main configuration file is displayed, included files are not resolved. To resolve included files and display the entire configuration, use the `syslog-ng-ctl config --preprocessed` command.

Starting with AxoSyslog version 4.2, you can display the configuration identifier (if set) and the SHA256 has of the output of the `syslog-ng-ctl config --preprocessed` command by running `syslog-ng-ctl config --id`. For details, see Configuration identifier.

The `config` 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.

- `--id` or `-i`

  Display the configuration identifier.

- `--preprocessed` or `-p`

  Resolve the included files and display the entire configuration.

- `--verify` or `-v`

  Verify the configuration that AxoSyslog is currently running.

List referenced files

You can use the `syslog-ng-ctl list-files` command to list files referenced in your configuration, for example, certificates or external configuration files. Available in AxoSyslog 3.23.1 and later.

Export the configuration graph

`syslog-ng-ctl export-config-graph`

Use the `syslog-ng-ctl export-config-graph` command to export the graph of the configuration that AxoSyslog is currently running.

Reloading the configuration

`syslog-ng-ctl reload [options]`

Use the `syslog-ng-ctl reload` command to reload the configuration file of AxoSyslog without having to restart the AxoSyslog application. The `syslog-ng-ctl reload` works like a SIGHUP.

The `syslog-ng-ctl reload` command returns 0 if the operation was successful, 1 otherwise.

Reopening destination files

`syslog-ng-ctl reopen`

Use the `syslog-ng-ctl reopen` command to make AxoSyslog close and reopen its destination files. This is useful after an external log rotation tool has renamed or removed the files that AxoSyslog writes.

Stopping AxoSyslog

`syslog-ng-ctl stop [options]`

Use the `syslog-ng-ctl stop` command to stop the AxoSyslog process.

The `stop` 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.

- `--force` or `-f`

  Available in AxoSyslog 4.28 and later.

  Terminate AxoSyslog without waiting for the worker threads to finish.

  > **CAUTION:**
  >
  > Stopping AxoSyslog this way can result in message loss, because the worker threads do not get the chance to flush the messages they are processing.

Displaying license information

`syslog-ng-ctl show-license-info [options]`

Use the `syslog-ng-ctl show-license-info` command to display information about the license of the running AxoSyslog instance.

The `show-license-info` 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.

- `--json` or `-J`

  Display the license information in JSON format.

The healthcheck command

Available in AxoSyslog 4.2 and later.

You can use the `syslog-ng-ctl healthcheck` command to query the healthcheck status of AxoSyslog. The following health values are reported:

- `mainloop_io_worker_roundtrip_latency_nanoseconds` : mainloop->io-worker-job->mainloop roundtrip - a basic latency measure for AxoSyslog.
- `io_worker_latency_nanoseconds` : io-worker-job start latency.
- `syslogng_internal_events_queue_usage_ratio` : If you are using the `internal()` source in your configuration, then this value shows the saturation of the internal source's queue, ranging from 0 to 1. Non-zero values indicate some kind of disruption in the pipelines.

You can run `syslog-ng-ctl healthcheck --timeout <seconds>` to use as a boolean healthy/unhealthy check.

Health checks are also published as periodically updated metrics. You can configure the frequency of these checks with the `stats(healthcheck-freq())` option. The default is 5 minutes.

The attach command

Available in AxoSyslog 4.9 and later.

Connect to the standard IO (stdin, stdout, stderr) and display the results. Note that there can only be one attached process at a time.

`syslog-ng-ctl attach [attach-mode] [options]`

The `syslog-ng-ctl attach` command has the following parameters:

- Attach mode: `logs` , `debugger` , or `stdio` . If you do not set the attach mode, `syslog-ng-ctl` uses `stdio` .
    - Use `logs` to access the internal log messages of `syslog-ng` . For example, the following command changes the log level to `trace` and accesses the internal logs of `syslog-ng` :

```
syslog-ng-ctl attach logs --seconds 10 --log-level trace
```

- Use `debugger` to attach to the interactive debugger of `syslog-ng`. For example:

```
syslog-ng-ctl attach debugger
```

- Use `stdio` to display the output of the `syslog-ng` process. For example:

```
syslog-ng-ctl attach stdio --seconds 10
```

- Change `log-level` to the specified value:

  Higher log-levels automatically include messages from lower log-levels:

  - `default`: Just normal log messages.
  - `verbose`: Normal and verbose log messages.
  - `debug`: Include debug messages of AxoSyslog.
  - `trace`: Include trace messages of how messages are processed.
- How long to attach to the process: `--seconds`. For example:

```
syslog-ng-ctl attach stdio --seconds 10
```

Files

`/opt/syslog-ng/sbin/syslog-ng-ctl`

See also

syslog-ng.conf.5

syslog-ng.8

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site.
- For news and notifications about AxoSyslog, visit the Axoflow blog.

- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.4 The dqtool tool manual page

Name

`dqtool` – Display the contents of a disk-buffer file created with AxoSyslog.

Synopsis

```
dqtool [command] [options]
```

Description

> **Note**
>
> **The `dqtool` application is distributed with the AxoSyslog system logging application, and is usually part of the AxoSyslog package.**

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, for example, `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: The cat command

```
./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
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPAD
DPADDPADDPADDPADDPADD
Mar  3 10:52:05 tristram localprg[1234]: seq: 0000011631, runid: 1267609923, stamp:
2010-03-03T10:52:05
PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPAD
DPADDPADDPADDPADDPADD
```

The info command

`info [DISK-BUFFER_FILE]`

Shows status information about the specified disk-buffer file. For example:

- Empty, normal disk-buffer file

```
/opt/syslog-ng/bin/dqtool info /opt/syslog-ng/var/syslog-ng-
00000.qf/var/lib/syslog-ng/syslog-ng-00000.qf
```

Example output:

```
Disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-
00000.qf/var/lib/syslog-ng/syslog-ng-00000.qf', number_of_messages='0'
```

- Non-empty, reliable disk-buffer queue file

```
/opt/syslog-ng/bin/dqtool info /opt/syslog-ng/var/syslog-ng-00000.rqf
```

```
Reliable disk-buffer state loaded; filename='/opt/syslog-ng/var/syslog-ng-
00000.rqf', number_of_messages='10'
```

The relocate command

`relocate [options] [files]`

Use the `relocate` command to move or rename disk-buffer (also called disk-queue) files. Note that this option modifies the persist file. Stop AxoSyslog before using this command.

The `relocate` command has the following options:

- `--all` or `-a`

  Relocate every disk-buffer file that is listed in the AxoSyslog persist file.

- `--new_path` or `-n`

  The directory where you want to move the disk-bufffer files. For example: `/var/disk-buffers`

- `--persist` or `-p`

The path to the AxoSyslog persist file. The `relocate` command automatically updates the entries of the disk-buffer files in the persist file.

Examples

Relocate a single queue file:

```
bin/dqtool relocate --new_path /tmp/dq --persist var/syslog-ng.persist /tmp/syslog-ng-00000.rqf
```

Relocate multiple queue files:

```
bin/dqtool relocate --new_path /tmp/dq --persist var/syslog-ng.persist /tmp/syslog-ng-00000.rqf /tmp/syslog-ng-00001.rqf
```

Relocate every queue file:

```
bin/dqtool relocate --new_path /tmp/dq --persist var/syslog-ng.persist --all
```

The assign command

`assign [options] [file]`

Use the `assign` command to assign an existing disk-buffer file to a destination, by adding it to the AxoSyslog persist file under a specific persist name. This is useful when the persist file has no entry for the disk-buffer file, and you want to attach the file to a destination manually. Note that this command modifies the persist file. Stop AxoSyslog before using it.

If the disk-buffer file is specified by filename only, `dqtool` appends it to the current working directory. You must know the persist name that your destination uses.

The `assign` command has the following options:

- `--example` or `-e`

  Print usage examples and exit.

- `--persist=<persist-file>` or `-p`

  The path to the AxoSyslog persist file to update.

- `--persist_name=<persist-name>` or `-n`

  The persist name to assign the disk-buffer file to. If the persist file already contains an entry with this name, `dqtool` overwrites it and prints the previous value.

Example: The assign command

```
bin/dqtool assign -p var/syslog-ng.persist \
                  -n "afsocket_dd_qfile(stream,localhost:15554)" \
                  /tmp/syslog-ng-dq/syslog-ng-00000.rqf
```

The truncate command

`truncate [options] [files]`

Use the `truncate` command to reclaim the unused space of abandoned disk-buffer files.

> **CAUTION:**
>
> Truncating a disk-buffer file that AxoSyslog is actively using can result in data loss. Stop AxoSyslog, or make sure that the files are no longer in use, before you run this command.

The `truncate` command has the following options:

- `--force` or `-f`

  Perform the truncation. Without this option, `dqtool` only prints a warning and exits without modifying any file.

Example: The truncate command

```
bin/dqtool truncate --force /tmp/syslog-ng-00000.rqf
```

The output looks like:

```
Disk-buffer /tmp/syslog-ng-00000.rqf has been truncated, reclaimed space: 1.453125
GiB
```

Files

`/opt/syslog-ng/bin/dqtool`

See also

syslog-ng.conf.5

syslog-ng.8

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.5 The loggen manual page

Name

`loggen` – Generate syslog messages at a specified rate

Note: If you don't have AxoSyslog installed and want to run `loggen` from a container, you can use:

```
docker run --rm -ti --entrypoint "" ghcr.io/axoflow/axosyslog /usr/bin/loggen
[options] target [port]
```

Synopsis

`loggen [options] target [port]`

Description

> **Note**
>
> **The `loggen` application is distributed with the AxoSyslog system logging application, and is usually part of the AxoSyslog package.**

The `loggen` application is a 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` : The average rate of the sent messages 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.

- `--client-port`

  Available in AxoSyslog 4.14 and later.

  Use the specified outbound port to connect to the server. Usable only for a single connection (when `--active-connections` is 1 and `--idle-connections` is not set).

- `--debug`

  Enable debug messages of `loggen` .

- `--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.

- `--perf`

  Disables rate limiting to send as many messages as possible.

- `--permanent` or `-T`

  Keep sending logs indefinitely, without time limit.

- `--proxied [<version>]` or `-H [<version>]`

  Prepend a PROXY protocol header to the generated messages. The optional argument selects the version of the PROXY protocol: `1` or `2`. Default value: `1`

Use the `--proxy-src-ip`, `--proxy-dst-ip`, `--proxy-src-port`, and `--proxy-dst-port` options to set the addresses in the header.

- `--proxied-tls-passthrough`

  Send the PROXY protocol header before the encrypted payload. Use it together with the `--proxied` and `--use-ssl` options.

- `--proxy-dst-ip <ip-address>`

  The destination IP address to set in the PROXY protocol header.

- `--proxy-dst-port <port>`

  The destination port to set in the PROXY protocol header.

- `--proxy-src-ip <ip-address>`

  The source IP address to set in the PROXY protocol header.

- `--proxy-src-port <port>`

  The source port to set in the PROXY protocol header.

- `--quiet` or `-Q`

  Display statistics only when `loggen` is finished. If not set, the statistics are displayed every second.

- `--rate <message/second>` or `-r <message/second>`

  The number of messages generated per second for every active connection. Default value: 1000. To disable rate limiting, use the `--perf` flag.

  If you want to change the message rate while loggen is running, send SIGUSR1 to double the message rate, or SIGUSR2 to halve it:

  ```
  kill -USR1 <loggen-pid>
  ```

  ```
  kill -USR2 <loggen-pid>
  ```

- `--rate-burst-start`

  Start sending messages at the full rate set in the `--rate` option, instead of ramping the rate up gradually. Useful for testing rate limiting.

- `--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 -`

- `--reconnect`

  Try to reconnect when destination connections are lost.

- `--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.

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 --stream --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 --stream --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 --stream --read-file -
```

Files

`/opt/syslog-ng/bin/loggen`

See also

syslog-ng.conf.5

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.6 The pdbtool manual page

Name

`pdbtool` – An application to test and convert pattern database rules

Synopsis

```
pdbtool [command] [options]
```

Description

The AxoSyslog 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, AxoSyslog 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.

Global options

The following options are available for every `pdbtool` command.

- `--debug` or `-d`

Enable debug/diagnostic messages on `stderr`. Note that in the `patternize` command, the `-d` short form sets the `--delimiters` option instead.

- `--module <module>`

  Load the specified module.

- `--module-path <path>`

  The colon-separated list of directories to search for modules.

- `--trace` or `-t`

  Enable trace messages on `stderr`.

- `--verbose` or `-v`

  Enable verbose messages on `stderr`.

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 AxoSyslog and it might also help to track down pattern-matching problems. The dump utility can dump the tree used for matching the PROGRAM or the MSG parts.

- `--debug` or `-d`

  Enable debug/diagnostic messages on `stderr`.

- `--pdb` or `-p`

  Name of the pattern database file to use.

- `--program` or `-P`

  Displays the RADIX tree built from the patterns belonging to the `${PROGRAM}` application.

- `--program-tree` or `-T`

  Display the `${PROGRAM}` tree.

- `--verbose` or `-v`

  Enable verbose messages on `stderr`.

Example and sample output:

```
pdbtool dump -p patterndb.xml  -P 'sshd'
```

```
'p'
     'assword for'
    @QSTRING:@
      'from'
        @QSTRING:@
          'port '
        @NUMBER:@ rule_id='fc49054e-75fd-11dd-9bba-001e6806451b'
          ' ssh' rule_id='fc55cf86-75fd-11dd-9bba-001e6806451b'
            '2' rule_id='fc4b7982-75fd-11dd-9bba-001e6806451b'
     'ublickey for'
    @QSTRING:@
      'from'
        @QSTRING:@
          'port '
        @NUMBER:@ rule_id='fc4d377c-75fd-11dd-9bba-001e6806451b'
          ' ssh' rule_id='fc5441ac-75fd-11dd-9bba-001e6806451b'
            '2' rule_id='fc44a9fe-75fd-11dd-9bba-001e6806451b'
```

The match command

`match [options]`

Use the `match` command to test the rules in a pattern database. The command tries to match the specified message against the patterns of the database, evaluates the parsers of the pattern, and also displays which part of the message was parsed successfully. The command returns with a `0` (success) or `1` (no match) return code and displays the following information:

- the class assigned to the message (that is, system, violation, and so on),
- the ID of the rule that matched the message, and
- the values of the parsers (if there were parsers in the matching pattern).

The `match` command has the following options:

- `--color-out` or `-c`

  Color the terminal output to highlight the part of the message that was successfully parsed.

- `--debug` or `-d`

  Enable debug/diagnostic messages on `stderr`.

- `--debug-csv` or `-C`

  Print the debugging information returned by the `--debug-pattern` option as comma-separated values.

- `--debug-pattern` or `-D`

  Print debugging information about the pattern matching. See also the `--debug-csv` option.

- `--file=<filename-with-path>` or `-f`

  Process the messages of the specified log file with the pattern database. This option allows to classify messages offline, and to apply the pattern database to already existing logfiles. To read the messages from the standard input (`stdin`), specify a hyphen (`-`) character instead of a filename.

- `--filter=<filter-expression>` or `-F`

  Print only messages matching the specified AxoSyslog 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`

  An AxoSyslog template expression that is used to format the output messages.

- `--verbose` or `-v`

  Enable verbose messages on `stderr`.

Example

The following command checks if the `patterndb.xml` file recognizes the `Accepted publickey for myuser from 127.0.0.1 port 59357 ssh6` message:

```
pdbtool match -p patterndb.xml -P sshd -M "Accepted publickey for myuser from
127.0.0.1 port 59357 ssh6"
```

Example

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 AxoSyslog documentation 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.

- `--sort` or `-s`

  Sort files into alphabetic order during the merge (first sort by filename, then by directory name).

- `--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 AxoSyslog documentation . The `patternize` command is available only in version 3.2 and later.

- `--debug`

Enable debug/diagnostic messages on stderr. Note that unlike in the other commands, the `-d` short form is not available here, because it sets the `--delimiters` option.

- `--delimiters=<delimiters>` or `-d`

  The set of characters used to tokenize the log messages. Default value: `:&~?![]=,;()'"`

- `--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`

  Include a generated name in the parsers, for example, `.dict.string1` , `.dict.string2` , and so on. Default value: `no`

- `--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>`

  The number of example log messages to include in the pattern database for every pattern. Default value: `1`

- `--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 AxoSyslog 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

syslog-ng.conf.5

syslog-ng.8

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.7 The persist-tool manual page

Name

`persist-tool` – Display the content of the persist file

Synopsis

```
persist-tool [command] [options]
```

Description

The `persist-tool` application is a utility that can be used to dump the content of the persist file, and manipulate its content.

WARNING: Persist-tool is a special tool for AxoSyslog experts. Do not use the tool unless you know exactly what you are doing. Misconfiguring it will result in irrecoverable damage to the persist file, without any warning.

Limitations

Wildcard characters are not supported in file/directory names.

The dump command

Use the `dump` command to print the current content of the persist file in JSON format to the console.

```
dump &lt;options&gt; &lt;persist_file&gt;
```

Example: `persist-tool dump /opt/syslog-ng/var/syslog-ng.persist`

The output looks like:

```
run_id = { "value": "00 00 00 00 0C 00 00 00 " }
host_id = { "value": "00 00 00 00 5F 49 2F 01 " }
```

The `dump` command has the following options:

- `--help` or `-?`

  Display a brief help message.

The add command

`add <options> <input_file>`

Use the `add` command to add or modify a specified state-entry in the persist file. The state-entry should be in the same format as the `dump` command displays it. If the given state-entry already exists, it will be updated. Otherwise, a new value will be added. If the given persist state is invalid, it will be skipped.

To use the `add` command: use `persist-tool dump` to print the content of the current persist file, and redirect it to a file. Edit the content of this file. Use `persist-tool add` with this file to modify the persist. The `add` command has the following options:

- `--help` or `-?`

  Display a brief help message.

- `--output-dir=<directory>` or `-o`

  Required parameter. The directory where the persist file is located at. Unless you also set the `--persist-name` option, the name of the persist file stored in this directory must be `syslog-ng.persist`.

- `--persist-name=<persist_name>` or `-p`

  The name of the persist file to modify. If you do not set this option, `persist-tool` uses the default `syslog-ng.persist` filename.

Example: `/opt/syslog-ng/bin/persist-tool add dump_persist -o .`

The valid output looks like:

```
log_reader_curpos(Application)      OK
affile_sd_curpos(/var/aaa.txt)        OK
```

The invalid output looks like:

```
log_reader_curpos(Application)      OK
wrong
FAILED (error: Invalid entry syntax)
affile_sd_curpos(/var/aaa.txt)        OK
```

The generate command

`generate <options>`

Use the `generate` command to create an empty persist file. The generated file is always named `syslog-ng.persist`.

The `generate` command has the following options:

- `--force` or `-f`

  Overwrite the persist file if it already exists.

  > **CAUTION:**
  >
  > Use this option with care: `persist-tool` does not ask for confirmation, and the contents of the overwritten persist file cannot be recovered.

- `--output-dir=<directory>` or `-o`

  The directory to generate the persist file into.

Example: `/opt/syslog-ng/bin/persist-tool generate -o /opt/syslog-ng/var`

Files

`/opt/syslog-ng/bin/persist-tool`

See also

syslog-ng.conf.5

syslog-ng.8

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.8 The secure-logging manual page

Name

`secure-logging` – Forward integrity and confidentiality for system logs.

Synopsis

```
$(slog --key-file &lt;host key file&gt; --mac-file &lt;MAC file&gt; $RAWMSG)
```

Description

Secure logging is an extension to AxoSyslog providing forward integrity and confidentiality of system logs. It is implemented in form of a module and is configured as a template in the AxoSyslog configuration file.

The main objective of the secure logging module is to provide tamper evident logging, i.e. to adequately protect log records of an information system and to provide a sensor indicating attack attempts. The secure logging module achieves this by authentically encrypting each log record with an individual cryptographic key used only once and protecting the integrity of the whole log archive by a cryptographic authentication code. Each attempt to tamper with either an individual log record or the log archive itself will be immediately detected during log archive verification. Therefore, an attacker can no longer tamper with log records without being detected.

In order to use the log file created by the secure logging module for analysis, the log file must first be decrypted and its integrity verified. This is achieved with a command line utility that is part of the secure logging module and is installed as part of the AxoSyslog package. This utility can easily be integrated into the import module of existing analysis environments.

The secure logging environment uses a cryptographic key for encrypting log entries. Each individual log entry is encrypted with its own key which is immediately discarded after successful encryption in order to provide forward integrity. An efficient algorithm generates the key for the next log entry based the key used for encrypting the previous log entry. The resulting chain of keys preserves forward integrity, i.e. a potential attacker cannot deduce the previous key from the current key.

In order to ease log verification and analysis, a sequence number is added to each log entry. This sequence number is also added to the encryption key creating a one-to-one correspondence of the encryption key with the log entry. In order to prevent truncation

attacks or deletion of individual log entries, a message authentication code (MAC) is iteratively applied to the complete log archive and stored in a separate file. It is used upon verification to check the integrity of the whole log archive.

Here is an example of three short original log messages that will be sent to a destination with secure logging enabled:

```
This is a log message
And here comes another log message
This is a log message with a longer text that is processed without any problems
```

In order to inspect the status of the secure logging environment, one can check the sequence counter by querying the key file with the `slogkey` utility like this:

```
user@host:~> slogkey --counter /etc/syslog-ng/host.key
counter=3
```

Counting starts at zero. This is why the counter is set to three after processing three messages.

The output of the secure logging template for the three example messages is shown below. One clearly sees the sequence number that was prepended to each message. The colon indicates the end of the sequence number and the start of the original message. As three message were processed, the sequence counter of the key will be three.

```
AAAAAAAAAAA=:k3/dYpLsgO2tUJKSauo6dycIBzW6OTC3pyA9TP+7AnqFgEojBzgC2rcK4OPfRtr8yg==
AQAAAAAAAAA=:smw0ATISVgN+BYEu5d7OLBE7aQhHpK9Ro4MndmNgSVrqhcmRCBCj6DUnD6ku0Z29CKJ0N6
LAJUgByX4Ev+g=
AgAAAAAAAAA=:5UVybnKL1EAbgC4CLfd8HpgurjREf4LEN61/yWHSD2hbXjRD4QmQdtbwguT1chzdItKSQA
Sps9QRIvR5Jd4AHzHfqxI4aRgdUBcNbAq26nwUCg5vPWygjmbtQaxZgCJYkry8slxCigmbTVs=
```

The output of a successful verification run is shown below.

```
0000000000000000: This is a log message
0000000000000001: And here comes another log message
0000000000000002: This is a log message with a longer text that is processed
without any problems
```

The original log messages have been successfully restored. Additionally, the sequence counter is also prepended to the clear text messages. This helps when in analyzing problems with a particular log entry. As real log files will contain thousands of entries. the sequence counter eases identification of erroneous entries.

Before the secure logging module can be used as part of an existing AxoSyslog installation, several preparatory activities need to be performed.

Key generation

In order to bootstrap the system, an initial key k0 must be created and installed on the log host before secure logging environment is started for the first time.

The newly created host key k0 has its counter set to 0 indicating that it represents the initial host key k0. This host key k0 must be kept secret and not be disclosed to third parties. It will be required to successfully decrypt and verify log archives processed by the secure logging environment. As each log entry will be encrypted with its own key, a new host key will be created after successful processing of a log entry and will replace the previous key. Therefore, the initial host key needs to be stored in a safe place before starting the secure logging environment, as it will be deleted from the log host after processing of the first log entry. The following steps must be done before starting the secure logging environment. Steps 1 and 2 are performed with the `slogkey` utility. See the `slogkey.1` manual page for details on how to generate a master key and to derive a host key from it. Step 3 and 4 depend on the actual deployment in a target environment.

1. Create a master key
2. Derive an initial host key k0 from a previously created master key1. Store the initial host key k0 in a safe location outside of the log host
3. Deploy the key k0 on the log host where the secure logging module will be used

Configuration

Secure logging is configured by adding the corresponding statements to the `syslog-ng.conf` file.

Secure logging is implemented as a template and is configured accordingly. Apart from the actual template configuration, no other settings are required in order to activate secure logging. The secure logging is activated by placing the following statement in the configuration file.

```
template("$(slog --key-file  &lt;host key file&gt; --mac-file &lt;MAC file&gt;
$RAWMSG)\n");
```

where:

- `slog`

  The name of the secure logging template function. This name can be also be found by calling `syslog-ng` with the `--module-registry` arguments and checking the `template-func` property of the secure logging module in the corresponding output.

- `--key-file` or `-k`

  The host key. `<host key file>` is the full path of the file storing the host key on the log host. If this arguments is not supplied or does not point to a valid regular key file, `syslog-ng` will not start and a display a corresponding error message.

- `--mac-file` or `-m`

  The MAC file. `<MAC file>` is the full path of the MAC file on the log host. The file does not need to exist, as it will be automatically created upon the initial start. If the path is not correct, `syslog-ng` will not start and a display a corresponding error message.

- `$RAWMSG`

  `$RAWMSG` provides access to the original log message received at the source. This macro is only available if the `store-raw-message` flag was set for the source. Otherwise, an empty string will be passed to the secure logging template. If access to the original message is not available, for example, if the source does not support the `store-raw-message` flag, then the `$MSG` macro can also be used. In this case, however, the integrity guarantee provided by secure logging is limited to the content that this macro provides and does not protect the complete original message.

- `\n`

  `\n` is the line separator. This is important, as the secure logging template expects log entries to be separated by a line separator. When detecting a line separator, the log entry is regarded as complete and is encrypted with the current host key.

Therefore, only a single line separator is allowed.

The secure logging template can be combined with any source or destination within the following limitations:

- Sources must be line-oriented. Secure logging uses a line separator in order to distinguish between individual log entries. Sources which provide data in a different format, for example, in the form of raw data obtained directly from a database system, cannot currently be used with the secure logging template, as the separation of log entries is not clearly defined for this type of data.
- Only sources for which the `store-raw-message` flag is implemented and set do benefit from the integrity guarantee provided by the secure logging template. Secure logging aims at protecting the integrity of complete log messages including all associated meta-data, such as timestamps and host names. `syslog-ng` parses the log message into its internal format and provide easy access to parts of a message through macros. While this is convenient when rewriting log messages, it is not helpful for secure logging. `syslog-ng` provides the `store-raw-message` flag which provides access to a copy of the original log message after parsing. This is the log message processed and protected by the secure logging template. If the source does not support the `store-raw-message` flag, then the $MSG macro can also be used. However, in this case the integrity guarantee provided by secure logging is limited to the content that this macro provides.
- Log rotation of any kind cannot be used with destinations using secure logging. The reason is that log rotate will overwrite, i.e. delete previous log files. This destroys the cryptographic chain of trust of the log entries making recovery impossible. In order to allow for a an efficient handling of log files, the secure logging environment features iterative verification. Using iterative verification, a log file can be verified in steps. For this to work, the log file must first be downloaded from the log host, together with the corresponding host key and MAC file to a verification host. After this download the log file can be safely deleted from the log host. Verification is then performed on the verification host using the iterative mode of the `slogverify` utility.

The following example configuration shows the use of the secure logging template on a file destination.

```
##############################################
# Minimal syslog-ng.conf file with secure logging enabled. Encrypted log
# entries will be logged to a single file called /var/log/messages.slog
#

@version: 4.4
```

```
@include "scl.conf"

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

source s_network {
    network(
        transport("udp")
        port(514)
        flags(store-raw-message)
    );
};

# Secure logging template definition
template secure_logging {
    template("$(slog --key-file /etc/syslog-ng/host.key --mac-file /etc/syslog-
ng/mac.dat $RAWMSG)\n");
};

# This configures a secure logging destination
destination d_local {
    file("/var/log/messages.slog" template(secure_logging));
};

log {
    source(s_local);

    # This source has the raw message flag set
    source(s_network);

    # This statement activates secure logging for this destination
    destination(d_local);
};
```

Log Verification

In order to analyze the log file created in a secure logging environment, the log files must be decrypted and their integrity be verified. Verification requires both the initial host key k0 and the corresponding MAC file and is performed with the `slogverify` utility. It is not

normally performed on the log host where the secure logging environment is producing log data. In a typical deployment, log files would be retrieved from the log host and transferred to a central log collector where verification it performed. As verification requires the use of the initial host key k0, it should only be performed in a trusted environment.

Normal mode

In normal mode, a complete log archive is verified at once. In a typical environment, this would mean retrieving a log file together with is MAC file from a log host and retrieving the corresponding initial key k0 form a safe location and supplying them to the `slogverify` utility. A typical call sequence for verification in normal mode would look like this:

```
slogverify --key-file host0.key --mac-file mac.dat /var/log/messages.slog
/var/log/verified/messages
```

where:

- `host0.key`

  The initial host key k0. Supplying the initial key k0 is enough for decrypting all log entries, as the key derivation algorithm is able to generate the necessary keys for all subsequent log entries based on the initial key k0.

- `mac.dat`

  The MAC file from the log host.

- `/var/log/messages.slog`

  The file containing the encrypted log entries as retrieved from a log host.

- `/var/log/verified/messages`

  The file receiving the plain text log after decryption.

Log files may become large and not fit into system memory. Verification is therefore performed in chunks. Each part of the log file is transferred to an internal buffer on which verification is performed. After the buffer has been processed, the next chunk is fetched. An optional buffer argument can be supplied to the `slogverify` utility in order to change the default buffer size of 1000 log entries to a number suitable for the system on which the verification is performed, for example:

```
slogverify --key-file host.key --mac-file mac.dat /var/log/messages.slog
/var/log/verified/messages 8000
```

See `slogverify` for details on verification in normal mode.

Iterative mode

Verification in normal mode may not be suitable for some application scenarios. Many log hosts use log rotation in order to preserve storage space. In log rotation, a threshold for the maximum amount of storage space and the number of generations is defined for different type of log files. When either storage space is exhausted or the number of generations is reached, the oldest log file will be overwritten by new incoming log data. This procedure is not possible in secure logging environment, as overwriting, i.e. deleting a log file would break the cryptographic chain that is established between the log entries. This comes as no surprise, as one of the main objectives of secure logging is to protect against deletion of log entries by a potential attacker.

In order allow for a procedure similar to log rotation, the secure logging environment features an iterative mode. In iterative mode, log files can be split into different files and each of these files can be verified separately. Care must be taken when performing verification in iterative mode, as each of the different log files needs to be accompanied by a copy of the host key and the MAC files present on the system at the time of retrieval. A typical usage scenario for the iterative mode would look like this:

1. Define a storage threshold for the secure logging file destination. In this example we assume 500MB.
2. Let the secure logging environment produce log data that is written to the destination until 500MB are reached.
3. Stop the secure logging environment and retrieve the log file, the host key and the MAC files from the log host.
4. Delete the log file on the log host but leave host key and MAC file untouched.
5. Restart the secure logging environment.
6. Perform verification in iterative mode with the log file, the host key and the MAC just retrieved.

Steps 2-6 have to repeated each time the log file reaches a size of 50 MB. Assuming that the log file parts will be named after the iteration, e.g. log.1, log.2, log.3, etc. and a similar convention is applied to the host keys and MAC files, a typical call sequence for the validation of a log file part in iterative mode after three iterations will look like this:

```
slogverify --iterative --prev-key-file host.key.2 --prev-mac-file mac.dat.2 --mac-file mac.dat
/var/log/messages.slog.3 /var/log/verified/messages.3
```

where:

- `host.key.2`

  The host key from the previous iteration. In this example, this is the second iteration.

- `mac.dat.2`

  The MAC file from the previous iteration. In the example, verification is performed during the third iteration, so the MAC file from the second iteration is required.

- `mac.dat`

  The current MAC file from the log host.

- `/var/log/messages.slog.3`

  The file part containing the encrypted log entries as retrieved from the log host during the third iteration.

- `/var/log/verified/messages.3`

  The file receiving the plain text log after decryption during the third iteration.

In a real deployment, the above steps would typically be automated using a scripting engine. See `slogverify` (1) for details on verification in iterative mode.

Files

`/usr/bin/slogkey`

`/usr/bin/slogencrypt`

`/usr/bin/slogverify`

`/etc/syslog-ng.conf`

See also

syslog-ng.conf.5

slogkey.1

slogencrypt.1

slogverify.1

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.9 The slogencrypt manual page

Name

`slogencrypt` – Encrypt existing plain text log files using the AxoSyslog secure logging environment.

Synopsis

```
slogencrypt [options] [arguments]
```

Description

The `slogencrypt` utility is used to encrypt plain text log file using an existing secure logging environment. Using this utility, log files obtained from a previous installation of `syslog-ng` or another logging system can be transferred to a secure logging environment. The order of the log entries is preserved. Encrypting plain text log files using an existing secure logging environment, requires the current encryption key to be supplied in order to preserve consistency.

General call sequence: `slogencrypt -k <key file> -m <MAC file> <new key file> <new MAC file> <plain text log> <output file> [counter]`

Arguments

- `<new key file>`

  The file that will contain the new current encryption key after successful encryption.

- `<new MAC file>`

  The file receiving the new current message authentication code (MAC) of the secure encrypted destination after encryption. In case an existing file is supplied, new entries will be appended.

- `<input log file>`

  The plain text log file that will be encrypted using the secure logging environment.

- `<output log file>`

The file that will contain the encrypted log entries from the supplied plain text log file after encryption.

- `counter`

  The current log entry counter of the secure encrypted destination after encryption. This is required if the log entries to be encrypted will be appended to an existing secure encrypted destination.

Options

- `--key-file` or `-k`

  The current host key from the system where the encryption will be performed.

- `--mac-file` or `-m`

  The current MAC file from the system where the encryption will be performed.

- `--help` or `-h`

  Display a help message.

Files

`/usr/bin/slogencrypt`

`/etc/syslog-ng.conf`

See also

syslog-ng.conf.5

syslog-ng.8

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.10 The slogkey manual page

Name

`slogkey` – Manage cryptographic keys for use with the AxoSyslog secure logging environment.

Synopsis

```
slogkey [options] [arguments]
```

Description

The `slogkey` utility is used to manage cryptographic keys for use with the secure logging module of AxoSyslog. Use this utility to create a master key, derive a host key to be used by a secure logging configuration and to display the current sequence counter of a key. The options determine the operating mode and are mutually exclusive.

Arguments

The arguments depend on the operating mode.

- Master key generation

  Call sequence: `slogkey --master-key <filename>`

  `<filename>` : The name of the file to which the master key will be written.

- Host key derivation

  Call sequence: `slogkey --derive-host-key <master key file> <host MAC address> <host serial number> <host key file>`

  - `<master key file>` : The master key from which the host key will be derived.
  - `<host MAC address>` : The MAC address of the host on which the key will be used. Instead of the MAC address, any other string that uniquely identifies a host can be supplied, e.g. the company inventory number.
  - `<host serial number>` : The serial number of the host on which the key will be used. Instead of the serial number, any other string that uniquely identifies a host can be supplied, for example, the company inventory number.

- `<host key file>` : The name of the file to which the host key will be written.

  NOTE: The newly created host key has its counter set to 0 indicating that it represents the initial host key k0. This host key must be kept secret and not be disclosed to third parties. It will be required to successfully decrypt and verify log archives processed by the secure logging environment. As each log entry will be encrypted with its own key, a new host key will be created after successful processing of a log entry and will replace the previous key. Therefore, the initial host key needs to be stored in a safe place before starting the secure logging environment, as it will be deleted from the log host after processing of the first log entry.

- Sequence counter display

  Call sequence: `slogkey --counter <host key file>`

  - `<host key file>` : The host key file from which the sequence will be read.

Options

- `--master-key` or `-m`

  Generates a new master key. `<filename>` is the name of the file storing the newly generated master key.

- `--derive-host-key` or `-d`

  Derive a host key using a previously generated master key.

- `--counter` or `-c`

  Display the current log sequence counter of a key.

- `--help` or `-h`

  Display a help message.

Files

`/usr/bin/slogkey`

`/etc/syslog-ng.conf`

See also

syslog-ng.conf.5

secure-logging.7

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.11 The slogverify manual page

Name

`slogverify` – Verify cryptographically secured logs.

Synopsis

```
slogverify [options] [arguments]
```

Description

The `slogverify` utility is used to verify the integrity of cryptographically secured logs and to decrypt log entries produced in a AxoSyslog secure logging environment.

- Normal mode: `slogverify -k <host key file> -m <input MAC file> <input file> <output file> [buffers]`
- Iterative mode: `slogverify -i -p <previous host key> -r <previous MAC> -m <current MAC> <input file> <output file> [buffers]`

Arguments

- `input file`

  An encrypted log file from the syslog-ng secure logging environment that will be verified.

- `output file`

  The file that will contain the plain text log entries after decryption and verification.

- `buffers`

  Optional number of input buffers. The number of buffers can be used for performance adjustments in case the log file to be verified is very large and cannot be processed at once. It is a positive number of log entries that can be held in memory during verification. The minimum number if 10 and the maximum number is 4294967295. If this argument is not supplied the default of 1000 is used.

Options

- `--iterative` or `-i`

  Iterative mode. This is useful in case the log files are periodically copied from the system on which they where generated to central collector. As log rotation, i.e. overwriting log files in order to preserve space cannot be done in a secure logging environment, the iterative mode can be used instead. This works as follows: If a certain storage limit is reached the log file together with the host key and the MAC file is copied to new destination and the old file is deleted. The verification is then performed in iterations, i.e. separately for each file that was retrieved from the log host. For this to work, it is important to always retrieve the corresponding host key and MAC files. The process can be automated, e.g. by calling slogverify in iterative mode from a script.<

- `--key-file` or `-k`

  The initial host key (k0). This option is used in normal mode only.

- `--mac-file` or `-m`

  The current MAC file used.

- `--prev-key-file` or `-p`

  The host key corresponding to the previous log file. This option can be used in iterative mode only. In theory, this can be initial host key (k0) but using this key might generate warnings, as the gap between the first log entry ever (log entry 0) and the first log entry of the current log file might be large.

- `--prev-mac-file` or `-r`

  The MAC file from the previous log file. This option can only be used in iterative mode.

- `--help` or `-h`

  Display a help message.

Files

`/usr/bin/slogverify`

`/etc/syslog-ng.conf`

See also

syslog-ng.conf.5

secure-logging.7

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

24.12 The syslog-debun manual page

Name

`syslog-debun` – `syslog-ng` DEBUg buNdle generator

Synopsis

```
syslog-debun [options]
```

Description

> **Note**
>
> **The `syslog-debun` application is distributed with the AxoSyslog system logging application, and is usually part of the AxoSyslog package.**

The `syslog-debun` tool collects and saves information about your AxoSyslog installation, making troubleshooting easier, especially if you ask help about your AxoSyslog 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 AxoSyslog 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`

- `-K`

  Include the private keys stored in the `/etc/syslog-ng` or `/opt/syslog-ng/etc` directory in the debug bundle.

  > **CAUTION:**
  >
  > The debug bundle is meant to be shared with support. Use this option only when the private keys are required to investigate your problem, and share the resulting bundle over a secure channel.

Debug mode options

- `-d`

  Start AxoSyslog 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 AxoSyslog in debug mode, using the specified command-line options. To exit debug mode, press Enter.

  - `-t <seconds>`

    Run AxoSyslog in non-interactive 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 AxoSyslog process. To trace system calls in debug mode, use both the `-s` and `-d` options.

Packet capture options

Capturing packets requires a packet capture tool on the host. The `syslog-debun` tool attempts to use `tcpdump` on most platforms, except for Solaris, where it uses `snoop` .

- `-i <interface>`

  Capture packets only on the specified interface, for example, `eth0` .

- `-p`

  Capture incoming packets using the following filter: `port 514 or port 601 or port 53`

- `-P <options>`

  Capture incoming packets using the specified filter.

- `-T <options>`

  Run `tcpdump` with the specified parameters instead of the default ones.

- `-t <seconds>`

  Run AxoSyslog in non-interactive 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 resolve info, and process tree.

```
syslog-ng-debun -r -d
```

Similar to `syslog-ng-debun -r`, but it also stops AxoSyslog, 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 AxoSyslog process.

```
syslog-ng-debun -r -d -s
```

Restart AxoSyslog in debug mode, and also trace the system calls (using `strace` or `truss`) of the AxoSyslog 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 AxoSyslog
- Start AxoSyslog in debug mode with default parameters (`-d`)
- Wait 10 seconds (`-t 10`)
- Stop AxoSyslog debugging
- Start AxoSyslog
- Stop packet capturing

Files

`/opt/syslog-ng/bin/syslog-ng-debun`

See also

syslog-ng.conf.5

Getting help

- The up-to-date documentation of AxoSyslog is available on the AxoSyslog documentation site .
- For news and notifications about AxoSyslog, visit the Axoflow blog .
- If you want to contact the developers directly to help with problems or report issues, contact us on Discord or GitHub .

This manual page is maintained by Axoflow

25 The syslog-ng OSE Documentation License

License terms for using, copying, and modifying documentation derived from the syslog-ng Open Source Edition documentation.

Permission is hereby granted, free of charge, to any person obtaining a copy of these documentation files (the “Documentation”), to use the Documentation subject to the following conditions:

1. The above copyright notice and this permission notice shall be included in all copies or portions of the Documentation. Any and all copies of the above copyright and this permission notice contained in the Documentation shall not be removed, obscured, or modified.

2. If you modify the Documentation, You must cause any modified files to carry prominent notices stating that You changed the files.

3. The Documentation contains trademarks and registered trademarks owned by One Identity and its affiliates. Permission is granted to make and distribute verbatim copies of the Documentation from the official syslog-ng.com site. Modified versions of the Documentation containing One Identity trademarks are permitted exclusively for internal use; however, all external distributions of modified Documentation are prohibited from containing any One Identity trademarks which include syslog-ng, the syslog-ng logo, Balabit, the Balabit logo, One Identity and the One Identity logo.

4. No relationship is established as a result of this Documentation license. Any statements that suggest a partnership or other relationship with One Identity and its affiliates are prohibited.

THE DOCUMENTATION IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENTATION OR THE USE OR OTHER DEALINGS IN THE DOCUMENTATION.

`YOU ACKNOWLEDGE THAT THE DOCUMENTATION YOU ARE DOWNLOADING IS SUBJECT TO THE RESTRICTIONS AND CONTROLS IMPOSED BY UNITED STATES EXPORT REGULATIONS.`

`YOU CERTIFY THAT:`

- YOU DO NOT INTEND TO USE THE DOCUMENTATION FOR ANY PURPOSE PROHIBITED BY UNITED STATES EXPORT REGULATIONS, INCLUDING, WITHOUT LIMITATION, TERRORISM, CYBER-ATTACKS, CYBER-CRIMES, MONEY-LAUNDERING, INDUSTRIAL ESPIONAGE, OR NUCLEAR, CHEMICAL OR BIOLOGICAL WEAPONS PROLIFERATION.
- YOU ARE NOT LISTED AS A DENIED PARTY ON ANY LIST GOVERNING UNITED STATES EXPORTS.
- YOU ARE NOT A NATIONAL OF ANY COUNTRY THAT IS NOT APPROVED TO RECEIVE A U.S. EXPORT. AS OF 2023, THESE COUNTRIES ARE CUBA, IRAN, NORTH KOREA, SYRIA, BELARUS AND RUSSIA.

If any of the above applicable conditions are not met or if your certification is inaccurate, this restricted copyright license is immediately terminated.