

The syslog-ng Open Source Edition 3.2 Administrator Guide

Second Edition

Publication date March 21, 2011

This manual is the primary documentation of the syslog-ng Open Source Edition 3.2 application.





The syslog-ng Open Source Edition 3.2 Administrator Guide

Product Marketing and Documentation Department

Revision History

For a list of documentation changes, see *Section 6.1, Summary of changes (p. xiv)*.

Copyright © 2006-2011 BalaBit IT Security Ltd.

This guide is published under the Creative Commons Attribution-Noncommercial-No Derivative Works (by-nc-nd) 3.0 license. The latest version is always available at <http://www.balabit.com/support/documentation>.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)

This documentation and the product it describes are considered protected by copyright according to the applicable laws.

The syslog-ng™ name and the syslog-ng™ logo are registered trademarks of BalaBit.

The BalaBit™ name and the BalaBit™ logo are registered trademarks of BalaBit.

Linux™ is a registered trademark of Linus Torvalds.

Debian™ is a registered trademark of Software in the Public Interest Inc.

Windows™ XP, 2003 Server, Vista, and 2008 Server are registered trademarks of Microsoft Corporation.

MySQL™ is a registered trademark of Oracle and/or its affiliates.

Oracle™, JD Edwards™, PeopleSoft™, and Siebel™ are registered trademarks of Oracle Corporation and/or its affiliates.

Red Hat™, Inc., Red Hat™ Enterprise Linux™ and Red Hat™ Linux™ are trademarks of Red Hat, Inc.

SUSE™ is a trademark of SUSE AG, a Novell business.

Solaris™ is a registered trademark of Sun Microsystems, Inc.

AIX™, AIX 5L™, AS/400™, BladeCenter™, eServer™, IBM™, the IBM™ logo, IBM System i™, IBM System i5™, IBM System x™, iSeries™, i5/OSTM, Netfinity™, NetServer™, OpenPower™, OS/400™, PartnerWorld™, POWER™, ServerGuide™, ServerProven™, and xSeries™ are trademarks or registered trademarks of International Business Machines.

Alliance Log Agent for System i™ is a registered trademark of Patrick Townsend & Associates, Inc.

All other product names mentioned herein are the trademarks of their respective owners.

Some rights reserved.

DISCLAIMER

BalaBit is not responsible for any third-party Web sites mentioned in this document. BalaBit does not endorse and is not responsible or liable for any content, advertising, products, or other material on or available from such sites or resources. BalaBit will not be responsible or liable for any damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through any such sites or resources.



Table of Contents

Preface	xi
1. Summary of contents	xi
2. Target audience and prerequisites	xii
3. Products covered in this guide	xii
4. Typographical conventions	xii
5. Contact and support information	xiii
5.1. Sales contact	xiii
5.2. Support contact	xiv
5.3. Training	xiv
6. About this document	xiv
6.1. Summary of changes	xiv
6.2. Feedback	xvi
6.3. Acknowledgments	xvi
1. Introduction to syslog-ng	1
1.1. What syslog-ng is	1
1.2. What syslog-ng is not	1
1.3. Why is syslog-ng needed?	2
1.4. What is new in syslog-ng Open Source Edition 3.2?	2
1.5. Who uses syslog-ng?	2
1.6. Supported platforms	3
2. The concepts of syslog-ng	4
2.1. The philosophy of syslog-ng	4
2.2. Logging with syslog-ng	4
2.2.1. The route of a log message in syslog-ng	4
2.3. Modes of operation	6
2.3.1. Client mode	6
2.3.2. Relay mode	7
2.3.3. Server mode	7
2.4. Global objects	7
2.5. Timezones and daylight saving	8
2.5.1. A note on timezones and timestamps	9
2.6. Stable and feature releases of syslog-ng OSE	9
2.7. The license of syslog-ng OSE	10
2.8. High availability support	10
2.9. The structure of a log message	10
2.9.1. BSD-syslog or legacy-syslog messages	10
2.9.2. IETF-syslog messages	12
3. Installing syslog-ng	16
3.1. Installing syslog-ng using the .run installer	16
3.1.1. Installing syslog-ng in client or relay mode	17
3.1.2. Installing syslog-ng in server mode	19
3.1.3. Installing syslog-ng without user-interaction	21
3.2. Installing syslog-ng on RPM-based platforms (Red Hat, SUSE, AIX)	22
3.3. Installing syslog-ng on Debian-based platforms	23
3.4. Compiling syslog-ng from source	23



3.5. Uninstalling syslog-ng	25
3.6. Configuring Microsoft SQL Server to accept logs from syslog-ng	26
4. The syslog-ng OSE quick-start guide	32
4.1. Configuring syslog-ng on client hosts	32
4.2. Configuring syslog-ng on server hosts	34
4.3. Configuring syslog-ng relays	35
4.3.1. Configuring syslog-ng on relay hosts	35
4.3.2. How relaying log messages works	36
5. The syslog-ng OSE configuration file	38
5.1. The syslog-ng configuration file	38
5.1.1. The configuration syntax in detail	38
5.1.2. Notes about the configuration syntax	40
5.2. Global and environmental variables	41
5.3. Managing large syslog-ng configurations	41
5.3.1. Including configuration files	41
5.3.2. Reusing configuration blocks	42
6. Collecting log messages — sources and source drivers	45
6.1. How sources work	45
6.2. Collecting internal messages	47
6.3. Collecting messages from text files	47
6.3.1. file() source options	48
6.4. Collecting messages from named pipes	52
6.4.1. pipe() source options	52
6.5. Collecting process accounting logs on Linux	56
6.5.1. pacct() options	56
6.6. Receiving messages from external applications	57
6.6.1. program() source options	57
6.7. Collecting messages on Sun Solaris	60
6.7.1. sun-streams() source options	61
6.8. Collecting messages using the IETF syslog protocol	64
6.8.1. syslog() source options	65
6.9. Collecting the system-specific log messages of a platform	71
6.10. Collecting messages from remote hosts using the BSD syslog protocol	72
6.10.1. tcp(), tcp6(), udp() and udp6() source options	74
6.11. Collecting messages from UNIX domain sockets	79
6.11.1. unix-stream() and unix-dgram() source options	80
7. Sending and storing log messages — destinations and destination drivers	85
7.1. Storing messages in plain-text files	86
7.1.1. file() destination options	87
7.2. Sending messages to named pipes	91
7.2.1. pipe() destination options	92
7.3. Sending messages to external applications	95
7.3.1. program() destination options	96
7.4. Storing messages in an SQL database	98
7.4.1. Using the sql() driver with an Oracle database	99
7.4.2. Using the sql() driver with a Microsoft SQL database	100
7.4.3. The way syslog-ng interacts with the database	101
7.4.4. sql() destination options	102



7.5. Sending messages to a remote logserver using the IETF-syslog protocol	106
7.5.1. syslog() destination options	107
7.6. Sending messages to a remote logserver using the legacy BSD-syslog protocol	112
7.6.1. tcp(), tcp6(), udp(), and udp6() destination options	112
7.7. Sending messages to UNIX domain sockets	117
7.7.1. unix-stream() and unix-dgram() destination options	117
7.8. Sending messages to a user terminal — usertty() destination	120
8. Routing messages: log paths and filters	121
8.1. Log paths	121
8.1.1. Embedded log statements	122
8.1.2. Log path flags	124
8.2. Managing incoming and outgoing messages with flow-control	125
8.2.1. Flow-control and multiple destinations	127
8.2.2. Configuring flow-control	127
8.3. Filters	129
8.3.1. Using filters	129
8.3.2. Combining filters with boolean operators	129
8.3.3. Comparing macro values in filters	130
8.3.4. Using wildcards, special characters, and regular expressions in filters	131
8.3.5. Tagging messages	132
8.3.6. Filter functions	132
8.4. Dropping messages	136
9. Global options of syslog-ng OSE	137
9.1. Configuring global syslog-ng options	137
9.2. Global options	137
10. TLS-encrypted message transfer	145
10.1. Secure logging using TLS	145
10.2. Encrypting log messages with TLS	146
10.2.1. Configuring TLS on the syslog-ng clients	146
10.2.2. Configuring TLS on the syslog-ng server	147
10.3. Mutual authentication using TLS	148
10.3.1. Configuring TLS on the syslog-ng clients	148
10.3.2. Configuring TLS on the syslog-ng server	150
10.4. TLS options	151
11. Manipulating messages	153
11.1. Customizing message format	153
11.1.1. Formatting messages, filenames, directories, and tablenames	153
11.1.2. Templates and macros	154
11.1.3. Macros of syslog-ng OSE	155
11.1.4. Using template functions	160
11.1.5. Template functions of syslog-ng OSE	160
11.2. Modifying messages	162
11.2.1. Conditional rewrites	163
11.3. Regular expressions	164
11.3.1. Types and options of regular expressions	165
11.3.2. Optimizing regular expressions	166
12. Parsing and segmenting structured messages	168
12.1. Parsing messages	168



12.2. Options of CSV parsers	170
13. Processing message content with a pattern database	173
13.1. Classifying log messages	173
13.1.1. The structure of the pattern database	174
13.1.2. How pattern matching works	175
13.1.3. Artificial ignorance	175
13.2. Using pattern databases	176
13.2.1. Using parser results in filters and templates	177
13.2.2. Downloading sample pattern databases	178
13.3. Correlating log messages	179
13.3.1. Referencing earlier messages of the context	180
13.4. Triggering actions for identified messages	180
13.5. Creating pattern databases	182
13.5.1. Using pattern parsers	182
13.5.2. What's new in the syslog-ng pattern database format V4	183
13.5.3. The syslog-ng pattern database format	184
14. Statistics of syslog-ng	191
15. Troubleshooting syslog-ng	193
15.1. Possible causes of losing log messages	193
15.2. Creating syslog-ng core files	194
15.3. Collecting debugging information with strace, truss, or tusc	194
15.4. Running a failure script	195
15.5. Stopping syslog-ng	195
16. Best practices and examples	196
16.1. General recommendations	196
16.2. Handling lots of parallel connections	196
16.3. Handling large message load	197
16.4. Using name resolution in syslog-ng	197
16.4.1. Resolving hostnames locally	198
16.5. Collecting logs from chroot	198
Appendix 1. The syslog-ng manual pages	200
syslog-ng	201
syslog-ng.conf	204
pdbtool	209
loggen	214
syslog-ng-ctl	216
Appendix 2. GNU Lesser General Public License	219
2.1. Preamble	219
2.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION	220
2.2.1. Section 0	220
2.2.2. Section 1	221
2.2.3. Section 2	221
2.2.4. Section 3	222
2.2.5. Section 4	222
2.2.6. Section 5	222
2.2.7. Section 6	223
2.2.8. Section 7	224



2.2.9. Section 8	224
2.2.10. Section 9	224
2.2.11. Section 10	224
2.2.12. Section 11	224
2.2.13. Section 12	225
2.2.14. Section 13	225
2.2.15. Section 14	225
2.2.16. NO WARRANTY Section 15	225
2.2.17. Section 16	226
2.3. How to Apply These Terms to Your New Libraries	226
Appendix 3. GNU General Public License	228
3.1. Preamble	228
3.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION	229
3.2.1. Section 0	229
3.2.2. Section 1	229
3.2.3. Section 2	229
3.2.4. Section 3	230
3.2.5. Section 4	230
3.2.6. Section 5	231
3.2.7. Section 6	231
3.2.8. Section 7	231
3.2.9. Section 8	231
3.2.10. Section 9	232
3.2.11. Section 10	232
3.2.12. NO WARRANTY Section 11	232
3.2.13. Section 12	232
3.3. How to Apply These Terms to Your New Programs	232
Appendix 4. Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License	234
Glossary	239
List of syslog-ng OSE parameters	243
Index	247



List of Examples

4.1. The default configuration file of syslog-ng OSE	33
4.2. A simple configuration for clients	33
4.3. A simple configuration for servers	35
4.4. A simple configuration for relays	36
5.1. A simple configuration file	38
5.2. Using required and optional parameters	39
5.3. Using global variables	41
5.4. Reusing configuration blocks	43
5.5. Defining blocks with multiple elements	43
5.6. Using arguments in blocks	44
6.1. A simple source statement	45
6.2. A source statement using two source drivers	45
6.3. Setting default priority and facility	45
6.4. Source statement on a Linux based operating system	46
6.5. Using the internal() driver	47
6.6. Using the file() driver	48
6.7. Tailing files	48
6.8. Using the pipe() driver	52
6.9. Using the program() driver	57
6.10. Using the sun-streams() driver	61
6.11. Using the syslog() driver	64
6.12. Using the udp() and tcp() drivers	73
6.13. Using the unix-stream() and unix-dgram() drivers	80
7.1. A simple destination statement	85
7.2. Using the file() driver	86
7.3. Using the file() driver with macros in the file name and a template for the message	86
7.4. Using the pipe() driver	92
7.5. Using the program() destination driver	95
7.6. Using the sql() driver	99
7.7. Using the sql() driver with an Oracle database	99
7.8. Using the sql() driver with an MSSQL database	100
7.9. Using SQL NULL values	105
7.10. Value: default	106
7.11. Using the syslog() driver	107
7.12. Using the tcp() driver	112
7.13. Using the unix-stream() driver	117
7.14. Using the userpty() driver	120
8.1. A simple log statement	121
8.2. Using embedded log paths	124
8.3. Using log path flags	125
8.4. Sizing parameters for flow-control	128
8.5. A simple filter statement	129
8.6. Comparing macro values in filters	131
8.7. Filtering with wildcards	132
8.8. Adding tags and filtering messages with tags	136



8.9. Skipping messages	136
9.1. Using global options	137
10.1. A destination statement using TLS	146
10.2. A source statement using TLS	147
10.3. Disabling mutual authentication	148
10.4. A destination statement using mutual authentication	149
10.5. A source statement using TLS	150
11.1. Using templates and macros	155
11.2. Using SDATA macros	158
11.3. Using the grep template function	161
11.4. Using pattern databases and the if template function	161
11.5. Using substitution rules	163
11.6. Setting message fields to a particular value	163
11.7. Using conditional rewriting	164
11.8. Using Posix regular expressions	165
11.9. Using PCRE regular expressions	166
11.10. Optimizing regular expressions in filters	166
12.1. Segmenting hostnames separated with a dash	168
12.2. Parsing Apache log files	169
12.3. Segmenting a part of a message	169
12.4. Adding the end of the message to the last column	171
13.1. Defining pattern databases	176
13.2. Using classification results	177
13.3. Using classification results for filtering messages	177
13.4. Using pattern parsers as macros	178
13.5. How syslog-ng OSE calculates <i>context-timeout</i>	180
13.6. Using message correlation	180
13.7. Generating messages for pattern database matches	181
13.8. Sending triggered messages to external applications	181
13.9. Pattern parser syntax	182
13.10. Using the STRING and ESTRING parsers	183
13.11. A V4 pattern database containing a single rule	189
1.1. Using required and optional parameters	205



List of Procedures

2.2.1. The route of a log message in syslog-ng	4
3.1.1. Installing syslog-ng in client or relay mode	17
3.1.2. Installing syslog-ng in server mode	19
3.2. Installing syslog-ng on RPM-based platforms (Red Hat, SUSE, AIX)	22
3.3. Installing syslog-ng on Debian-based platforms	23
3.4. Compiling syslog-ng from source	23
3.6. Configuring Microsoft SQL Server to accept logs from syslog-ng	26
4.1. Configuring syslog-ng on client hosts	32
4.2. Configuring syslog-ng on server hosts	34
4.3.1. Configuring syslog-ng on relay hosts	35
10.2.1. Configuring TLS on the syslog-ng clients	146
10.2.2. Configuring TLS on the syslog-ng server	147
10.3.1. Configuring TLS on the syslog-ng clients	148
10.3.2. Configuring TLS on the syslog-ng server	150
15.2. Creating syslog-ng core files	194
16.4.1. Resolving hostnames locally	198
16.5. Collecting logs from chroot	198



Preface

Welcome to the syslog-ng Open Source Edition 3.2 Administrator Guide!

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

1. Summary of contents

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

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

Chapter 3, Installing syslog-ng (p. 16) describes how to install syslog-ng OSE on various UNIX-based platforms using the precompiled binaries.

Chapter 4, The syslog-ng OSE quick-start guide (p. 32) provides a brief explanation of how to perform the most common log collecting tasks with syslog-ng OSE.

Chapter 5, The syslog-ng OSE configuration file (p. 38) discusses the configuration file format and syntax in detail, and explains how to manage large-scale configurations using included files and reusable configuration snippets.

Chapter 6, Collecting log messages — sources and source drivers (p. 45) explains how to collect and receive log messages from various sources.

Chapter 7, Sending and storing log messages — destinations and destination drivers (p. 85) describes the different methods to store and forward log messages.

Chapter 8, Routing messages: log paths and filters (p. 121) explains how to route and sort log messages, and how to use filters to select specific messages.

Chapter 9, Global options of syslog-ng OSE (p. 137) lists the global options of syslog-ng OSE and explains how to use them.

Chapter 10, TLS-encrypted message transfer (p. 145) shows how to secure and authenticate log transport using TLS encryption.

Chapter 11, Manipulating messages (p. 153) describes how to customize message format using templates and macros, how to rewrite and modify messages, and how to use regular expressions.

Chapter 12, Parsing and segmenting structured messages (p. 168) describes how to segment and process structured messages like comma-separated values.

Chapter 13, Processing message content with a pattern database (p. 173) explains how to identify and process log messages using a pattern database.

Chapter 14, Statistics of syslog-ng (p. 191) details the available statistics that syslog-ng OSE collects about the processed log messages.



Chapter 15, Troubleshooting syslog-ng (p. 193) offers tips to solving problems.

Chapter 16, Best practices and examples (p. 196) gives recommendations to configure special features of syslog-ng.

Appendix 1, The syslog-ng manual pages (p. 200) contains the manual pages of the syslog-ng OSE application.

Appendix 2, GNU Lesser General Public License (p. 219) includes the text of the LGPLv2.1 license applicable to the core of syslog-ng Open Source Edition.

Appendix 3, GNU General Public License (p. 228) includes the text of the GPLv2 license applicable to syslog-ng Open Source Edition.

Appendix 4, Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License (p. 234) includes the text of the Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License applicable to The syslog-ng Open Source Edition 3.2 Administrator Guide.

Glossary (p. 239) provides definitions of important terms used in this guide.

List of syslog-ng OSE parameters (p. 243) provides cross-references to the definitions of options, parameters, and macros available in syslog-ng OSE.

Index (p. 247) provides cross-references to important terms used in this guide.

2. Target audience and prerequisites

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

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

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

3. Products covered in this guide

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

- syslog-ng Open Source Edition (OSE) 3.2.0 and later

4. Typographical conventions

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



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

**Tip**

Tips provide best practices and recommendations.

**Note**

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

**Warning**

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

Command

Commands you have to execute.

Emphasis

Reference items, additional readings.

/path/to/file

File names.

Parameters

Parameter and attribute names.

Label

GUI output messages or dialog labels.

Menu

A submenu in the menu bar.

Button

Buttons in dialog windows.

5. Contact and support information

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

BalaBit IT Security Ltd.
1464 Budapest P.O. BOX 1279
Hungary
Tel: +36 1 371-0540
Fax: +36 1 208-0875
E-mail: info@balabit.com
Web: <http://www.balabit.com/>

5.1. Sales contact

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



5.2. Support contact

Subscribe to the mailing list of the syslog-ng community here.

To report bugs found in syslog-ng, *visit this page*.

Product support, including 7x24 online support is available in various packages. For support options, *visit the following page*.

Precompiled binary packages are *available for free for the supported Linux and BSD platforms here*. Visit *this link for the list of supported platforms*.

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

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

The *BalaBit Online Support System* offers 24 hours technical support. This system is available only for users with a valid support contract and a MyBalaBit account. To sign up for MyBalaBit, *visit this page*.

5.3. Training

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

6. About this document

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

The latest version of this document can be downloaded from the BalaBit website at *the BalaBit website*.

For news and update notifications about the syslog-ng documentation, visit *the BalaBit Documentation Blog*.

6.1. Summary of changes

6.1.1. Version 3.2 - 3.3

Changes in product:



Changes in documentation:

- Procedures have been restructured to facilitate easier understanding.
- Latin abbreviations have been replaced in document with their English equivalents.
- Links to sections in the document have been harmonized.
- Links to external web pages have been clarified.
- Added a note about the statistics of messages with high facility numbers to *Chapter 14, Statistics of syslog-ng* (p. 191).
- The description of the `dir_perm()` option of file destinations has been clarified.



- The description of the `time_reap()` option has been added to *Section 7.1.1, file() destination options (p. 87)*.
- The descriptions of facility and priority values used by the `internal()` source has been added to *Section 6.2, Collecting internal messages (p. 47)*.
- The description of the `pad_size()` option has been clarified in *Section 6.3.1, file() source options (p. 48)*.
- The description of the `port()` option has been added to *Section 7.4, Storing messages in an SQL database (p. 98)*.
- The working of the SQL destination driver has been clarified.
- The description of the `pad_size()` option has been added to *Section 7.1.1, file() destination options (p. 87)* and *Section 7.2.1, pipe() destination options (p. 92)*.
- The handling of IETF-syslog messages has been clarified in *Section 2.9.2, IETF-syslog messages (p. 12)*.
- Documented that multiple configuration files can be included from a directory in *Section 5.3.1, Including configuration files (p. 41)*.
- The syntax of the configuration file has been clarified in *Section 5.3.1, Including configuration files (p. 41)*.
- The `follow_freq()` option has been removed from *Section 6.11.1, unix-stream() and unix-dgram() source options (p. 80)*.
- The `optional()` option has been removed from *Section 6.8.1, syslog() source options (p. 65)*.
- The `ip_tos()`, `ip_ttl()`, `so_broadcast()`, `so_sndbuf()`, `follow_freq()` options have been removed from *Section 6.10.1, tcp(), tcp6(), udp() and udp6() source options (p. 74)*.
- The `so_broadcast()`, `so_sndbuf()` options have been removed from *Section 6.11.1, unix-stream() and unix-dgram() source options (p. 80)*.

6.1.2. Version 3.1 - 3.2

Changes in product:

- *Section 5.2, Global and environmental variables (p. 41)*
- *Section 5.3.2, Reusing configuration blocks (p. 42)*
- *Section 6.5, Collecting process accounting logs on Linux (p. 56)*
- *Section 6.9, Collecting the system-specific log messages of a platform (p. 71)*
- *Section 8.3.3, Comparing macro values in filters (p. 130)*
- *Section 11.1.4, Using template functions (p. 160)*
- *Section 11.2.1, Conditional rewrites (p. 163)*
- *Section 13.3, Correlating log messages (p. 179)*
- *Section 13.4, Triggering actions for identified messages (p. 180)*
- *Section 13.5.2, What's new in the syslog-ng pattern database format V4 (p. 183)*
- *pdbscript(1) (p. 209)*

Changes in documentation:

- The contents of the guide have been updated for syslog-ng Open Source Edition 3.2.



- The structure of the document has changed significantly; several important topics and feature descriptions were moved to their own chapters. These chapters contain most (if not all) the information related to the feature, including concepts, configuration, and reference.
- Many other clarifications, corrections, reordering of sections, and other changes too numerous to list.

6.2. Feedback

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

6.3. Acknowledgments

BalaBit would like to express its gratitude to the syslog-ng users and the syslog-ng community for their invaluable help and support, including the community members listed at [syslog-ng Community Page](#).



Chapter 1. Introduction to syslog-ng

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

1.1. What syslog-ng is

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

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

1.2. What syslog-ng is not

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



1.3. Why is syslog-**ng** needed?

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

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

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

For details on how you can use syslog-**ng** to comply with various regulations, see the *Regulatory compliance and system logging* whitepaper [available here](#)

1.4. What is new in syslog-**ng** Open Source Edition 3.2?

Version 3.2 of syslog-**ng** Open Source Edition includes the following main features:

- syslog-**ng** Open Source Edition 3.2 uses a new pattern database format dubbed V3 that has several improvements over the older V1 format.

1.5. Who uses syslog-**ng**?

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

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

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

- [Allianz Hungary Insurance Co.](#)
- [Narvisite Inc.](#)
- [Svenska Handelsbanken AB](#)
- [Swedish National Debt Office](#)



1.6. Supported platforms

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

- The source code of syslog-ng Open Source Edition is released under the GPLv2 license and is [available here](#).
- Precompiled binary packages provided by Balabit are [available for free for the supported Linux and BSD platforms here](#).
- For syslog-ng Open Source Edition packages for Solaris 8-10, [visit this page](#)
- For syslog-ng Open Source Edition packages for IBM AIX 5 and later, [visit this page](#)
- For syslog-ng Open Source Edition packages for HP-UX, [visit this page](#)
- For syslog-ng Open Source Edition packages for Mac OS X, [visit this page](#)
- Packages for routers running OpenWRT or a similar embedded Linux distribution are [available here](#)



Chapter 2. The concepts of syslog-ng

This chapter discusses the technical concepts of syslog-ng.

2.1. The philosophy of syslog-ng

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

2.2. Logging with syslog-ng

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

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

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

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

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

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

Purpose:

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

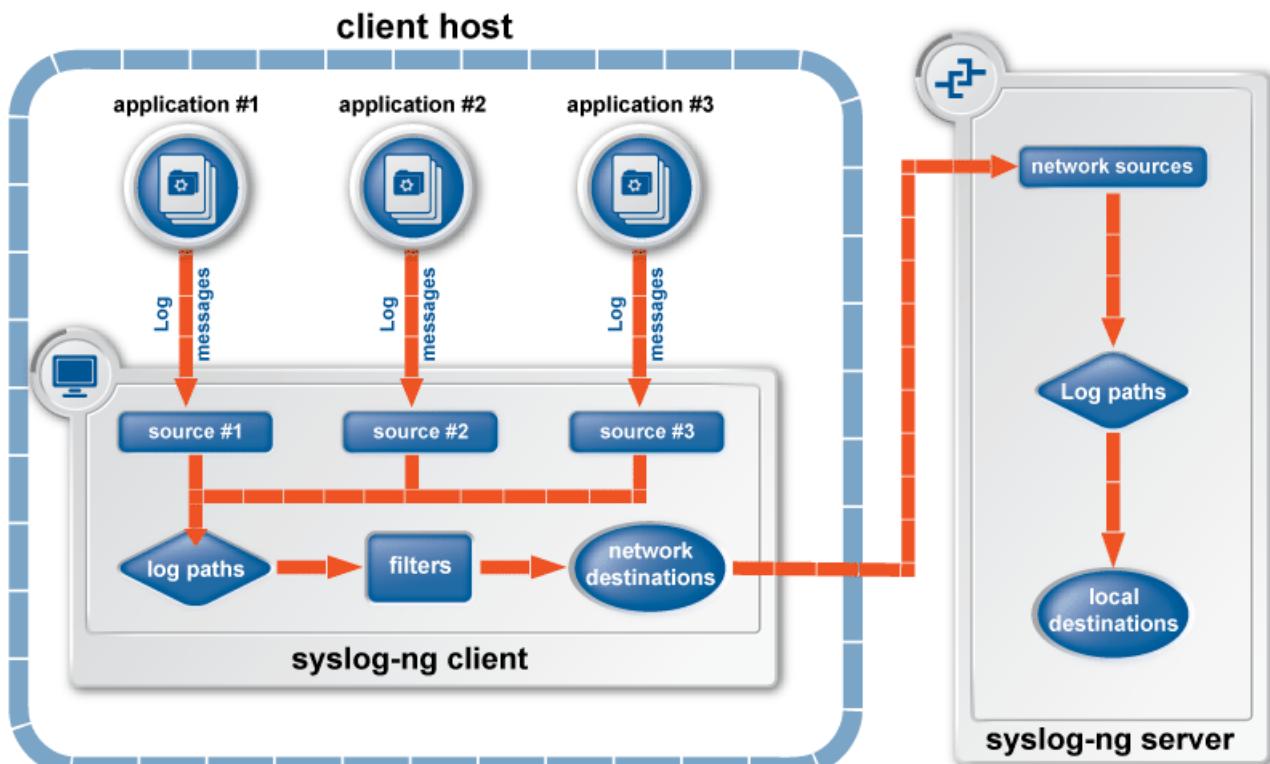


Figure 2.1. The route of a log message

Steps:

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



Warning

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



Note

The syslog-ng client sends a message to *all* matching destinations by default. As a result, a message may be sent to a destination more than once, if the destination is used in multiple log statements. To prevent such situations, use the `final` flag in the destination statements. For details, see *Table 8.1, Log statement flags (p. 124)*.



- Step 5. The syslog-ng client processes the next log statement that includes the `/var/log/apache` source, repeating Steps 3-4.
- Step 6. The message sent by the syslog-ng client arrives to a source set in the syslog-ng server.
- Step 7. The syslog-ng server reads the message from its source and processes the first log statement that includes that source.
- Step 8. The syslog-ng server performs optional operations (message filtering, parsing, and rewriting) on the message; for example, it compares the message to the filters of the log statement (if any). If the message complies with all filter rules, syslog-ng sends the message to the destinations set in the log statement.

**Warning**

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

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

**Note**

The syslog-ng application can stop reading messages from its sources if the destinations cannot process the sent messages. This feature is called flow-control and is detailed in *Section 8.2, Managing incoming and outgoing messages with flow-control (p. 125)*.

2.3. Modes of operation

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

2.3.1. Client mode

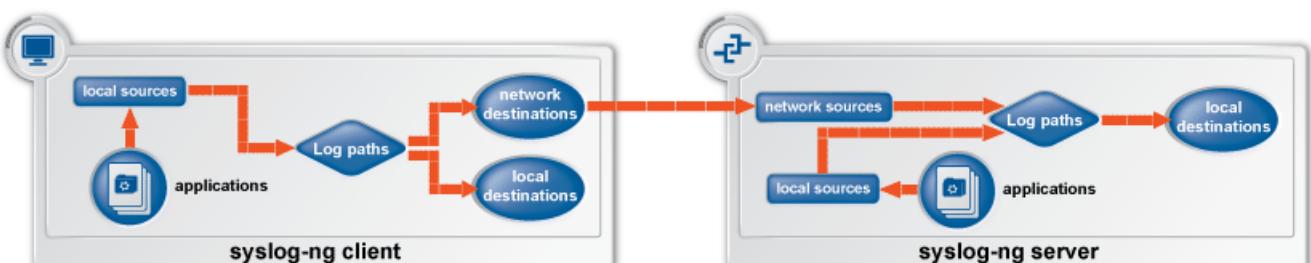


Figure 2.2. Client-mode operation

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



2.3.2. Relay mode

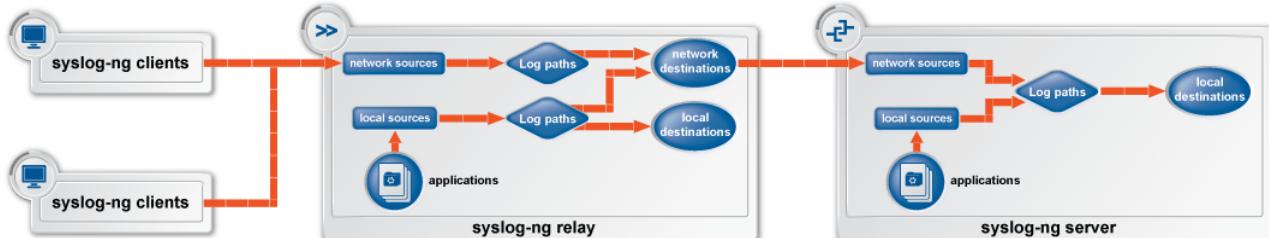


Figure 2.3. Relay-mode operation

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

2.3.3. Server mode

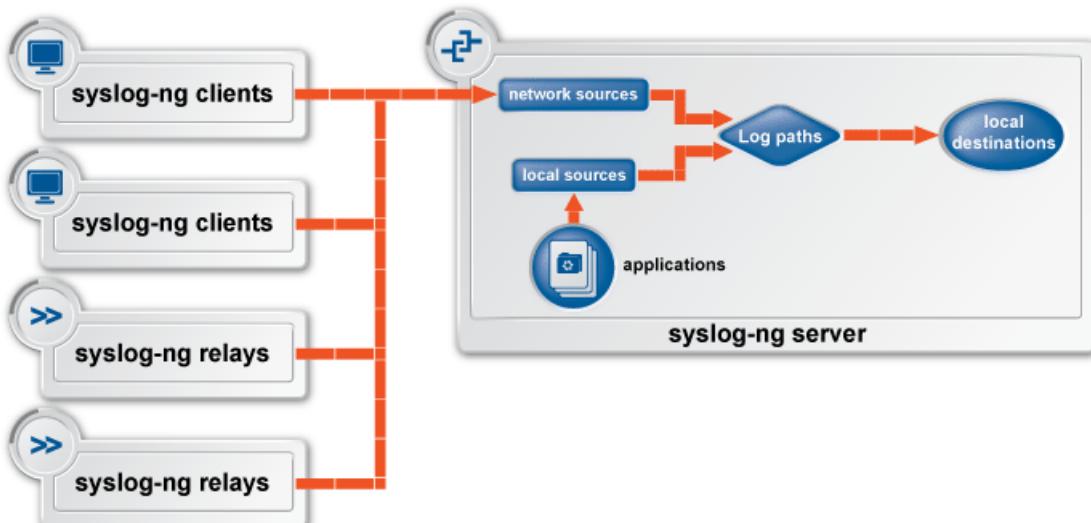


Figure 2.4. Server-mode operation

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

2.4. Global objects

The syslog-ng application uses the following objects:

- *Source driver*: A communication method used to receive log messages. For example, syslog-ng can receive messages from a remote host via TCP/IP, or read the messages of a local application from a file.
- *Source*: A named collection of configured source drivers.



- *Destination driver*: A communication method used to send log messages. For example, syslog-ng can send messages to a remote host via TCP/IP, or write the messages into a file or database.
- *Destination*: A named collection of configured destination drivers.
- *Filter*: An expression to select messages. For example, a simple filter can select the messages received from a specific host.
- *Macro*: An identifier that refers to a part of the log message. For example, the \$HOST macro returns the name of the host that sent the message. Macros are often used in templates and filenames.
- *Parser*: A rule that segments messages into separate columns at a predefined separator character (for example a comma). Every column has a unique name that can be used as a macro.
- *Rewrite rule*: A rule modifies a part of the message, for example, replaces a string, or sets a field to a specified value.
- *Log paths*: A combination of sources, destinations, and other objects like filters, parsers, and rewrite rules. The syslog-ng application sends messages arriving to the sources of the log paths to the defined destinations, and performs filtering, parsing, and rewriting of the messages. Log paths are also called log statements. Log statements can include other (embedded) log statements to create complex log paths.
- *Template*: A template is a set of macros that can be used to restructure log messages or automatically generate file names. For example, a template can add the hostname and the date to the beginning of every log message.
- *Option*: Options set global parameters of syslog-ng, like the parameters of name resolution and timezone handling.

For details on the above objects, see *Section 5.1.1, The configuration syntax in detail (p. 38)*.

2.5. Timezones and daylight saving

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

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

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

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



- Step 2. Specify the `time_zone()` parameter for the source driver that reads the message. This timezone will be associated with the messages only if no timezone is specified within the message itself. Each source defaults to the value of the `_recv_time_zone()` global option.
- Step 3. Specify the timezone in the destination driver using the `time_zone()` parameter. Each destination driver might have an associated timezone value; syslog-ng converts message timestamps to this timezone before sending the message to its destination (file or network socket). Each destination defaults to the value of the `_send_time_zone()` global option.

**Note**

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

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

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

2.5.1. A note on timezones and timestamps

If the clients run syslog-ng, then use the ISO timestamp, because it includes timezone information. That way you do not need to adjust the `recv_time_zone()` parameter of syslog-ng.

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

2.6. Stable and feature releases of syslog-ng OSE

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

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

**Note**

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

**Warning**

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

2.7. The license of syslog-ng OSE

Starting with version 3.2, the syslog-ng Open Source Edition application is licensed under a combined LGPL+GPL license. The core of syslog-ng OSE is licensed under the GNU Lesser General Public License Version 2.1 license, while the rest of the codebase is licensed under the GNU General Public License Version 2 license.

**Note**

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

For details about the LGPL and GPL licenses, see *Appendix 2, GNU Lesser General Public License (p. 219)* and *Appendix 3, GNU General Public License (p. 228)*, respectively.

2.8. High availability support

Multiple syslog-ng servers can be run in fail-over mode. The syslog-ng application does not include any internal support for this, as clustering support must be implemented on the operating system level. A tool that can be used to create UNIX clusters is Heartbeat (for details, see [this page](#)).

2.9. The structure of a log message

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

- The old standard described in RFC 3164 (also called the BSD-syslog or the legacy-syslog protocol): see *Section 2.9.1, BSD-syslog or legacy-syslog messages (p. 10)*
- The new standard described in RFC 5424 (also called the IETF-syslog protocol): see *Section 2.9.2, IETF-syslog messages (p. 12)*

2.9.1. BSD-syslog or legacy-syslog messages

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

- PRI
- HEADER
- MSG

The total message cannot be longer than 1024 bytes.



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

**Note**

The syslog-*ng* application supports longer messages as well. For details, see the *log_msg_size()* option in *Section 9.2, Global options (p. 137)*. However, it is not recommended to enable messages larger than the packet size when using UDP destinations.

2.9.1.1. The PRI message part

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

**Note**

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

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



Numerical Code	Facility
16-23	locally used facilities (local0-local7)

Table 2.1. syslog Message Facilities

The following table lists the severity values.

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

Table 2.2. syslog Message Severities

2.9.1.2. The HEADER message part

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

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



Note

The syslog-ng application supports other timestamp formats as well, like ISO, or the PIX extended format. For details, see the *ts_format()* option in *Section 9.2, Global options (p. 137)*.

2.9.1.3. The MSG message part

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

2.9.2. 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:

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

The message corresponds to the following format:

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

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

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

2.9.2.1. The PRI message part

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



Note

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

Numerical Code	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd



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

Table 2.3. *syslog Message Facilities*

The following table lists the severity values.

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

Table 2.4. *syslog Message Severities*

2.9.2.2. The HEADER message part

The HEADER part contains the following elements:

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



- *MESSAGEID*: The ID number of the message.

**Note**

The syslog-*ng* application supports other timestamp formats as well, like ISO, or the PIX extended format. The timestamp used in the IETF-syslog protocol is derived from RFC3339, which is based on ISO8601. For details, see the *ts_format()* option in *Section 9.2, Global options (p. 137)*.

The syslog-*ng* OSE application will truncate the following fields:

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

2.9.2.3. The STRUCTURED-DATA message part

The STRUCTURED-DATA message part may contain meta-information about the syslog message, or application-specific information such as traffic counters or IP addresses. STRUCTURED-DATA consists of data blocks enclosed in brackets (*[]*). Every block include the ID of the block, and one or more *name=value* pairs. The syslog-*ng* application automatically parses the STRUCTURED-DATA part of syslog messages, which can be referenced in macros (for details, see *Section 11.1.3, Macros of syslog-*ng* OSE (p. 155)*). An example STRUCTURED-DATA block looks like:

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

2.9.2.4. The MSG message part

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



Chapter 3. Installing syslog-ng

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

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



Note

For instructions on compiling syslog-ng Open Source Edition from the source code, see *Procedure 3.4, Compiling syslog-ng from source (p. 23)*.

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

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

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



Note

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

The syslog-ng application can be installed interactively following the on-screen instructions as described in *Section 3.1, Installing syslog-ng using the .run installer (p. 16)*, and also without user interaction using the silent installation option — see *Section 3.1.3, Installing syslog-ng without user-interaction (p. 21)*.

3.1. Installing syslog-ng using the .run installer

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

- To install syslog-ng on clients or relays, complete *Procedure 3.1.1, Installing syslog-ng in client or relay mode (p. 17)*.
- To install syslog-ng on your central logserver, complete *Procedure 3.1.2, Installing syslog-ng in server mode (p. 19)*.
- To install syslog-ng without any user-interaction, complete *Section 3.1.3, Installing syslog-ng without user-interaction (p. 21)*.

**Note**

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

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

Purpose:

Complete the following steps to install syslog-ng Open Source Edition on clients or relays. For details on the different operation modes of syslog-ng, see *Section 2.3, Modes of operation (p. 6)*.

Steps:

**Note**

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

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

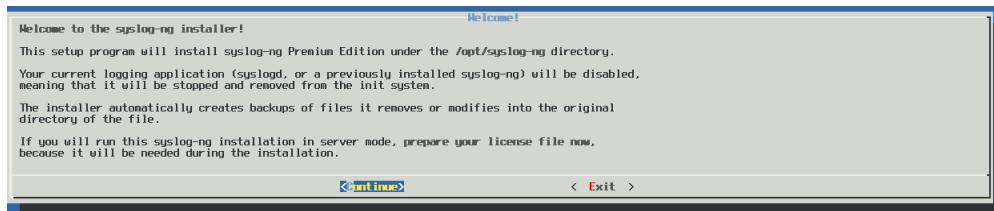


Figure 3.1. The welcome screen

- Step 2. *Accepting the EULA*: You can install syslog-ng only if you understand and accept the terms of the End-User License Agreement (EULA). The full text of the EULA can be displayed during installation by selecting the **Show EULA** option, and is also available in this guide for convenience at *Appendix 3, GNU General Public License (p. 228)*. Select **Accept** to accept the EULA and continue the installation. If you do not accept the terms of the EULA for some reason, select **Reject** to cancel installing syslog-ng.

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

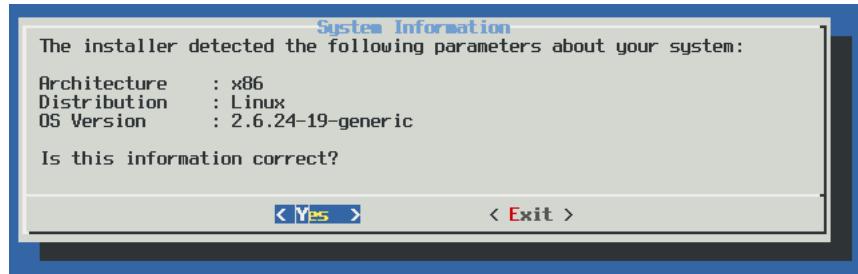


Figure 3.2. Platform detection

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

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

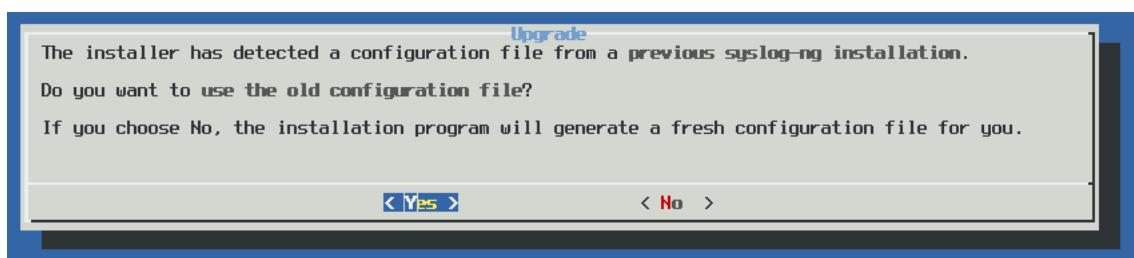


Figure 3.3. Upgrading syslog-ng

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

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

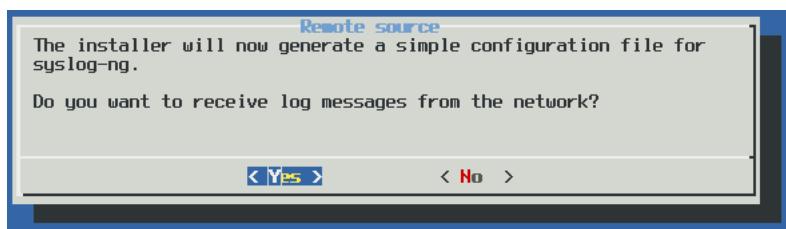


Figure 3.4. Accepting remote messages

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

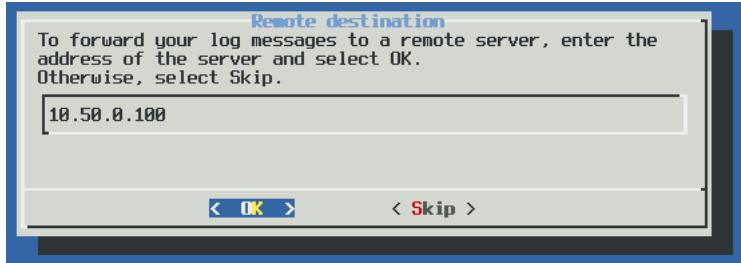


Figure 3.5. Forwarding messages to the logserver

**Note**

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

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

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

3.1.2. Procedure – Installing syslog-ng in server mode

Purpose:

Complete the following steps to install syslog-ng on logservers. For details on the different operation modes of syslog-ng, see *Section 2.3, Modes of operation (p. 6)*.

Steps:

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

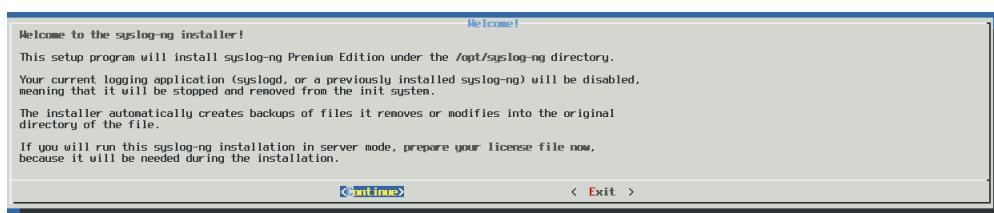


Figure 3.6. The welcome screen

- Step 2. *Accepting the EULA*: You can install syslog-ng only if you understand and accept the terms of the End-User License Agreement (EULA). The full text of the EULA can be displayed during installation by selecting



the **Show EULA** option, and is also available in this guide for convenience at *Appendix 3, GNU General Public License (p. 228)*. Select **Accept** to accept the EULA and continue the installation.

If you do not accept the terms of the EULA for some reason, select **Reject** to cancel installing syslog-ng.

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

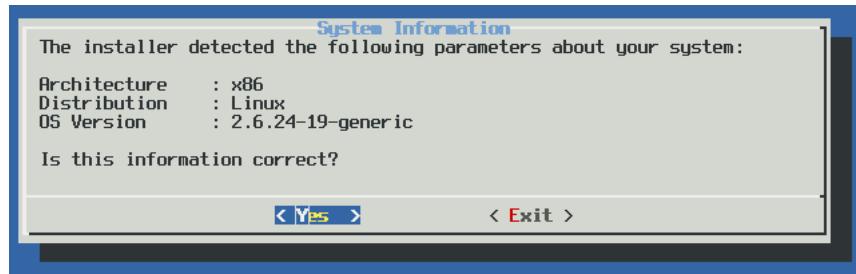


Figure 3.7. Platform detection

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

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

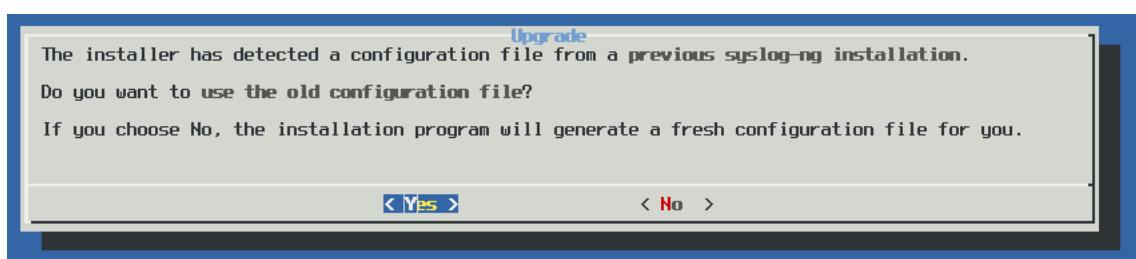


Figure 3.8. Upgrading syslog-ng

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

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



Figure 3.9. Accepting remote messages

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

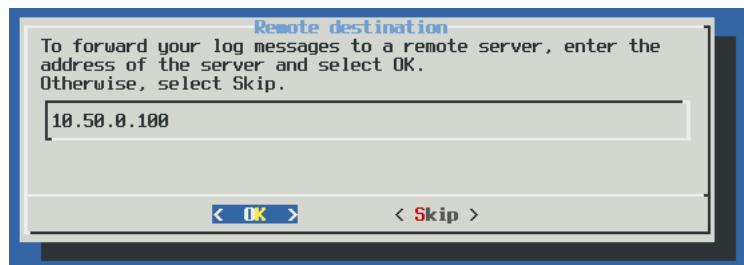


Figure 3.10. Forwarding messages to the logserver

**Note**

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

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

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

**Note**

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

3.1.3. Installing syslog-ng without user-interaction

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

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

**Warning**

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

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

- `--accept-eula` or `-a`: Accept the EULA.
- `--upgrade` | `-u`: Perform automatic upgrade — use the configuration file from an existing installation.
- `--remote <destination host>`: Send logs to the specified remote server. Not available when performing an upgrade.
- `--network`: Accept messages from the network. Not available when performing an upgrade.
- `--configuration <file>`: Use the specified configuration file.

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

Purpose:

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

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

Steps:

Step 1. Login to [your MyBalabit account](#) and download the [syslog-ng RPM package for your system](#).

Step 2. ■ If the host already uses syslog-ng for logging, execute the following command as root. Otherwise, skip this step.

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

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

**Note**

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

- Execute the following command as root:



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

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

Step 3.


Warning

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

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

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

Step 4. *Optional step for AIX systems:* To redirect the messages of the AIX Error log into syslog, create a file (for example /tmp/syslog-ng.add) with the following contents:

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

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

3.3. Procedure – Installing syslog-ng on Debian-based platforms

Purpose:

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

- Debian etch

Steps:

Step 1. Login to *your MyBalabit account* and download the *syslog-ng DEB package for your system*.

Step 2. Issue the following command as root:

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

Step 3. Answer the configuration questions of syslog-ng. These are described in detail in *Section 3.1, Installing syslog-ng using the .run installer (p. 16)*.

3.4. Procedure – Compiling syslog-ng from source

Purpose:

To compile syslog-ng Open Source Edition (OSE) from the source code, complete the following steps. Alternatively, you can use the precompiled binary packages. Precompiled binary packages are available for free for the supported Linux and BSD platforms *here*.

Steps:



- Step 1. Download the latest version of syslog-ng OSE [here](#). The source code is available as a tar.gz archive file.
- Step 2. Download the latest version of the EventLog library [here](#).
- Step 3. Install the following packages that are required to compile syslog-ng. These packages are available for most UNIX/Linux systems. Alternatively, you can also download the sources and compile them.
- the *gcc* C compiler (at least version 2.7.2),
 - the *GNU flex* lexical analyser generator, [available here](#);
 - the *bison* parser generator, [available here](#);
 - and the development files of the *glib* library, [available here](#).
- Step 4. If you want to use the spoof-source function of syslog-ng, install the development files of the *libnet* library, [available here](#).
- Step 5. If you want to use the */etc/hosts.deny* and */etc/hosts.allow* for TCP access, install the development files of the *libwrap* (also called TCP-wrappers) library, [available here](#).
- Step 6. Uncompress the eventlog archive using the

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

or the

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

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

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

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

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

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

- Step 9. Uncompress the syslog-ng archive using the

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

or the

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

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

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

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



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

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

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



Note

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



Warning

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

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

For information on configuring syslog-ng, see the *Chapter 4, The syslog-ng OSE quick-start guide (p. 32)*.

3.5. Uninstalling syslog-ng

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



- If you have installed syslog-ng using the .run installer: Execute the `uninstall.sh` script located at `/opt/syslog-ng/bin/uninstall.sh`. The uninstall script will automatically restore the syslog daemon used before installing syslog-ng. To completely remove syslog-ng, including the configuration files, use the `uninstall.sh --purge` command.
- If you have installed syslog-ng from a .deb package: Execute the `dpkg -r syslog-ng` command to remove syslog-ng; or the `dpkg -P syslog-ng` command to remove syslog-ng and the configuration files as well. Note that removing syslog-ng does not restore the syslog daemon used before syslog-ng.
- If you have installed syslog-ng from an .rpm package: Execute the `rpm -e syslog-ng` command to remove syslog-ng. Note that removing syslog-ng does not restore the syslog daemon used before syslog-ng.

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

Purpose:

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

Steps:

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

Step a.

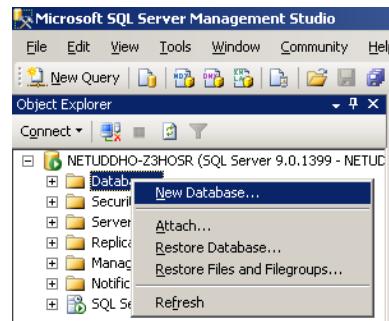


Figure 3.11. Creating a new MSSQL database 1.

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



Step b.

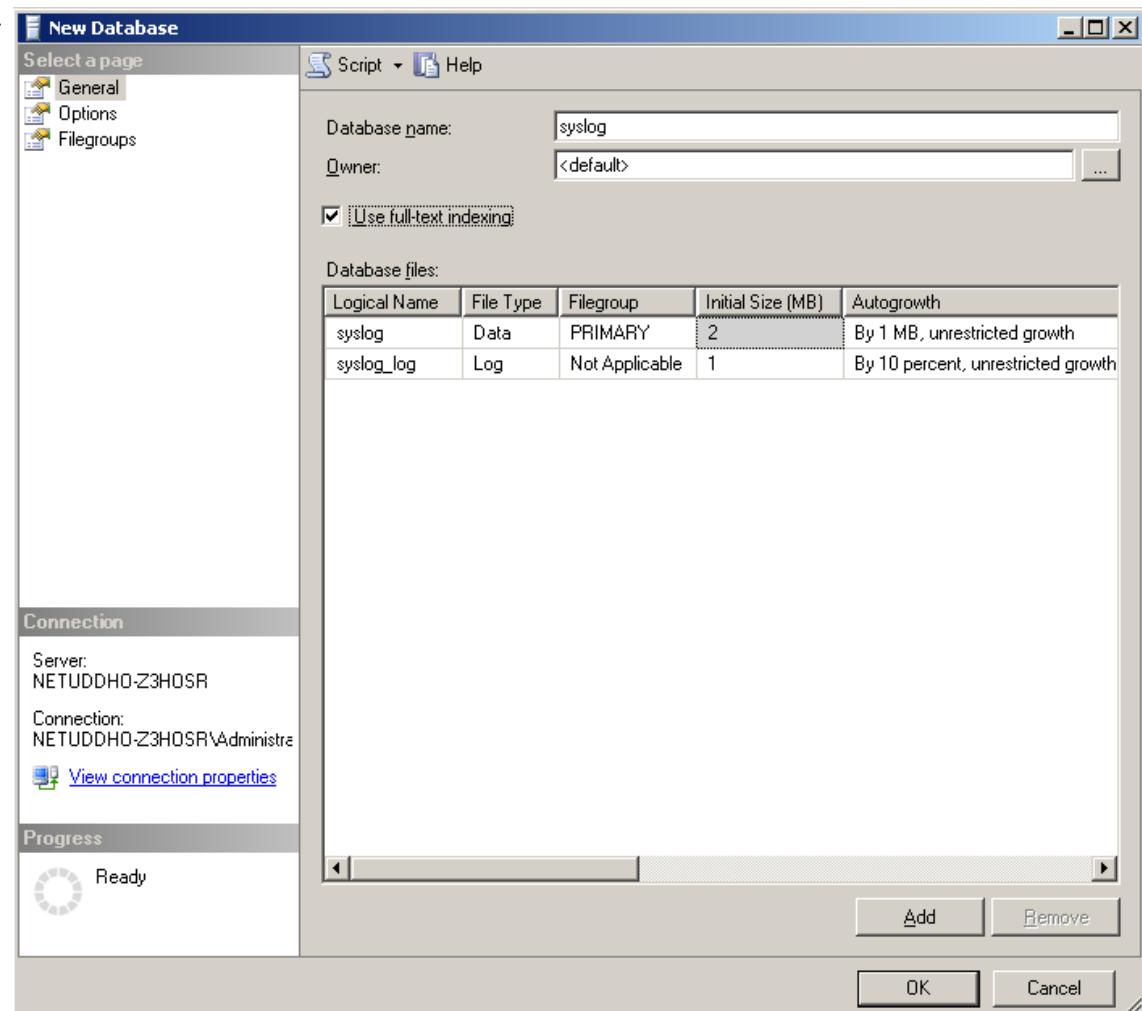


Figure 3.12. Creating a new MSSQL database 2.

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

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



Step a.

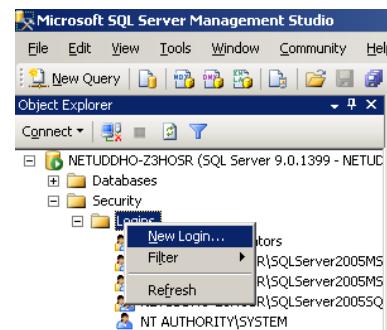


Figure 3.13. Creating a new MSSQL user 1.

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

Step b.

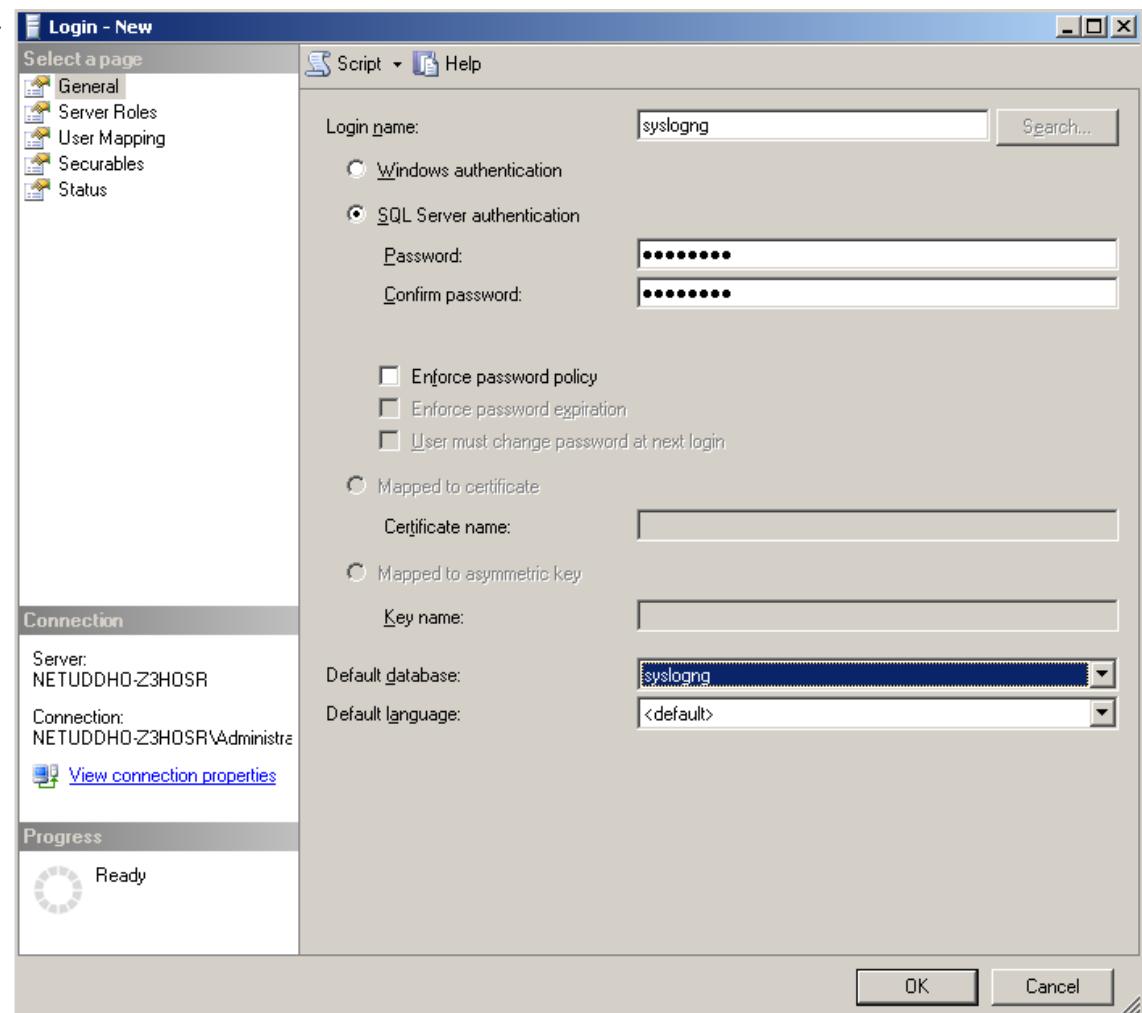


Figure 3.14. Creating a new MSSQL user 2.

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



- Step c. Select the **SQL Server Authentication** option and enter a password for the user.
- Step d. In the **Default database** field, select the database created in Step 2 (for example *syslogng*).
- Step e. In the **Default language** field, select the language of log messages that you want to store in the database, then click **OK**.

**Warning**

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

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



Step g.

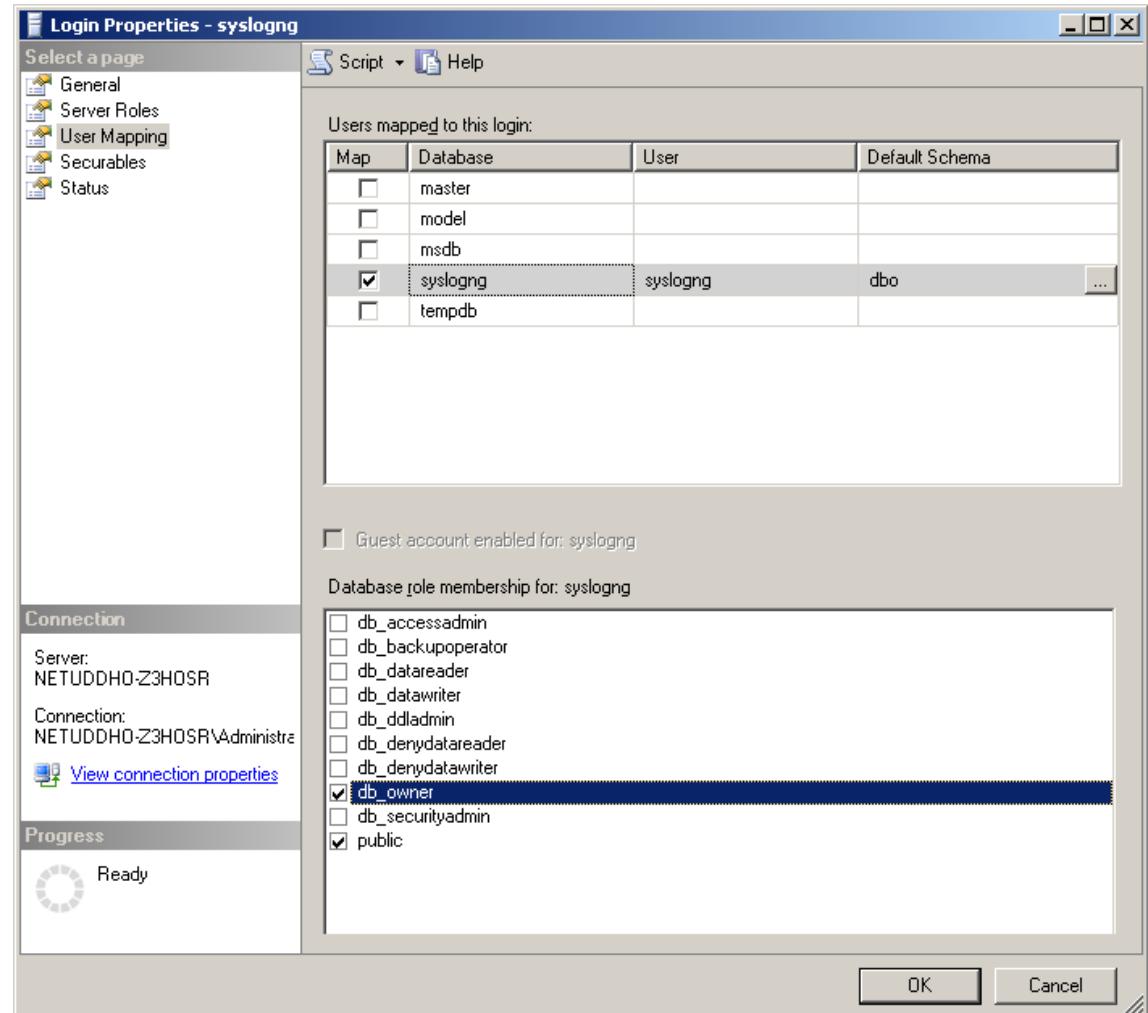


Figure 3.15. Associating database with the new user

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



Step 4.

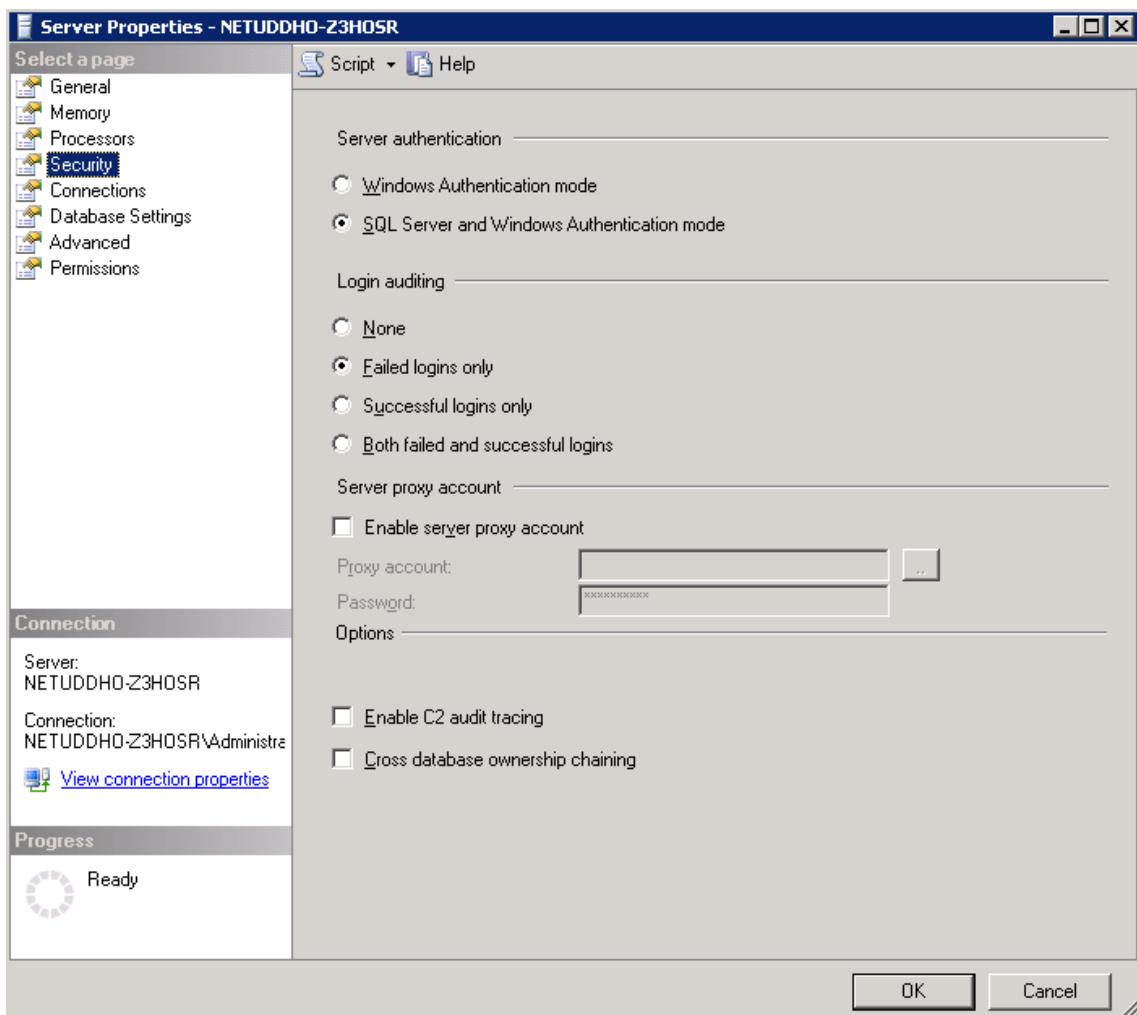


Figure 3.16. Associating database with the new user

Enable remote logins for SQL users.

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



Chapter 4. The syslog-ng OSE quick-start guide

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

4.1. Procedure – Configuring syslog-ng on client hosts

Purpose:

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

Steps:

- Step 1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see *Chapter 3, Installing syslog-ng (p. 16)*.
- Step 2. Configure the local sources to collect the log messages of the host. Starting with version 3.2, syslog-ng OSE automatically collects the log messages that use the native system logging method of the platform, for example, messages from `/dev/log` on Linux, or `/dev/klog` on FreeBSD. For a complete list of messages that are collected automatically, see *Section 6.9, Collecting the system-specific log messages of a platform (p. 71)*.

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 *Section 6.3, Collecting messages from text files (p. 47)*.



Tip

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



Note

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

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

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



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

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

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

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

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

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



Note

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

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

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



Note

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

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

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



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

The following is the default configuration file of syslog-ng OSE 3.2. It collects local log messages and the log messages of syslog-ng OSE and forwards them to a logserver using the IETF-syslog protocol.

```
@version: 3.2  
@include "scl.conf"  
source s_local { system(); internal(); };  
destination d_local {  
    file("/var/log/messages"); };  
log { source(s_local); destination(d_local); };
```



Example 4.2. A simple configuration for clients

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

```
@version: 3.2  
@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); };
```



4.2. Procedure – Configuring syslog-ng on server hosts

Purpose:

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

Steps:

- Step 1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see *Chapter 3, Installing syslog-ng (p. 16)*.
- Step 2. Starting with version 3.2, syslog-ng OSE automatically collects the log messages that use the native system logging method of the platform, for example, messages from `/dev/log` on Linux, or `/dev/klog` on FreeBSD. For a complete list of messages that are collected automatically, see *Section 6.9, Collecting the system-specific log messages of a platform (p. 71)*.
- Step 3. Configure the network sources that collect the log messages sent by the clients and relays. How the network sources should be configured depends also on the capabilities of your client hosts: many older networking devices support only the legacy BSD-syslog protocol (RFC3164) using UDP transport:

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

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

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

For other options, see *Section 6.8, Collecting messages using the IETF syslog protocol (p. 64)* and *Section 6.10, Collecting messages from remote hosts using the BSD syslog protocol (p. 72)*.



Note

Starting with syslog-ng OSE version 3.2, the `syslog()` source driver can handle both BSD-syslog (RFC 3164) and IETF-syslog (RFC 5424-28) messages.

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

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

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

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

For details on further macros and how to use them, see *Chapter 11, Manipulating messages (p. 153)*.

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

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



Step 6. Set filters, options (for example TLS encryption) and other advanced features as necessary.

**Note**

By default, the syslog-ng server will treat the relayed messages as if they were created by the relay host, not the host that originally sent them to the relay. In order to use the original hostname on the syslog-ng server, use the `keep_hostname(yes)` option both on the syslog-ng relay and the syslog-ng relay. This option can be set individually for every source if needed.

**Example 4.3. A simple configuration for servers**

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

```
@version: 3.2
@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); };
```

4.3. Configuring syslog-ng relays

4.3.1. Procedure – Configuring syslog-ng on relay hosts

Purpose:

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

Steps:

- Step 1. Install the syslog-ng application on the host. For details installing syslog-ng on specific operating systems, see *Chapter 3, Installing syslog-ng (p. 16)*.
- Step 2. Configure the network sources that collect the log messages sent by the clients.
- Step 3. Create a network destination that points to the syslog-ng server.
- Step 4. Create a log statement connecting the network sources to the syslog-ng server.
- Step 5. Configure the local sources that collect the log messages of the relay host.
- Step 6. Create a log statement connecting the local sources to the syslog-ng server.
- Step 7. Enable the `keep_hostname()` and disable the `chain_hostnames()` options.

**Note**

It is recommended to use these options on your syslog-ng OSE server as well.

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

**Note**

By default, the syslog-ng server will treat the relayed messages as if they were created by the relay host, not the host that originally sent them to the relay. In order to use the original hostname on the syslog-ng server, use the `keep_hostname(yes)` option both on the syslog-ng relay and the syslog-ng relay. This option can be set individually for every source if needed.

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

**Example 4.4. A simple configuration for relays**

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

```
@version: 3.2
@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);
};
```

4.3.2. How relaying log messages works

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

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

Many thanks to Lance Laursen for his excellent post about this topic on the [syslog-ng mailing list](#).



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

**Note**

It is also possible to resolve IP addresses locally, without relying on the DNS server. For details on local name resolution, see *Procedure 16.4.1, Resolving hostnames locally (p. 198)*.

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



Chapter 5. The syslog-ng OSE configuration file

5.1. The syslog-ng configuration file

The syslog-ng application is configured by editing the `syslog-ng.conf` file. Use any regular text editor application to modify the file. The location of the configuration file depends on how you installed syslog-ng OSE:

- **Binary packages downloaded from the Balabit website:** The `syslog-ng.conf` file is located under the `/opt/syslog-ng/etc/` directory. The precompiled syslog-ng OSE packages include sample configuration files as well. These packages are [available here](#).
- **Third-party packages:** Native packages of a platform (like the ones downloaded from Linux repositories) typically place the configuration file under the `/etc/syslog-ng/` directory.

5.1.1. The configuration syntax in detail

Every syslog-ng configuration file must begin with a line containing the version information of syslog-ng. For syslog-ng version 3.2, this line looks like:

```
@version: 3.2
```

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



Example 5.1. A simple configuration file

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

```
@version: 3.2

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

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

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

As a syslog-ng user described on a [mailing list](#):

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

—Alan McKinnon

- The main body of the configuration file consists of object definitions: sources, destinations, logpaths define which log message are received and where they are sent. All identifiers, option names and attributes,



and any other strings used in the syslog-*ng* configuration file are case sensitive. Objects must be defined before they are referenced in another statement. Object definitions (also called statements) have the following syntax:

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

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

**Tip**

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

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

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

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

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

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

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

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

For example, the following source statement has three options; the first two options (*file()* and *follow_freq()*) have a single parameter, while the third one (*flags()*) has two parameters:

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

Objects may have required and optional parameters. Required parameters are positional, meaning that they must be specified in a defined order. Optional parameters can be specified in any order using the *option(value)* format. If a parameter (optional or required) is not specified, its default value is used. The parameters and their default values are listed in the reference section of the particular object.

**Example 5.2. Using required and optional parameters**

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

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



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

- Some options are global options, or can be set globally, for example, whether syslog-ng OSE should use DNS resolution to resolve IP addresses. Global options are detailed in *Chapter 9, Global options of syslog-ng OSE* (p. 137).

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

- All identifiers, attributes, and any other strings used in the syslog-ng configuration file are case sensitive.
- Objects can be used before definition.
- To add comments to the configuration file, start a line with # and write your comments. These lines are ignored by syslog-ng.

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

**Tip**

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

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

5.1.2. Notes about the configuration syntax

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

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

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

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

- When enclosing object IDs (for example the name of a destination) between double-quotes ("mydestination"), the ID can include whitespace as well, for example:

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



- For notes on using regular expressions, see *Section 11.3, Regular expressions (p. 164)*.

5.2. Global and environmental variables

Starting with syslog-ng OSE version 3.2, it is possible to define global variables in the configuration file. Global variables are actually *name-value* pairs; when syslog-ng processes the configuration file during startup, it automatically replaces '*name*' with *value*. To define a global variable, use the following syntax:

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

The value can be any string, but special characters must be escaped. To use the variable, insert the name of the variable enclosed between backticks (`, similarly to using variables in Linux shells) anywhere in the configuration file.

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

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



Tip

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



Example 5.3. Using global variables

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

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

The syslog-ng OSE application will interpret this as:

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

5.3. Managing large syslog-ng configurations

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

5.3.1. Including configuration files

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



```
include "<filename>";
```

Where <filename> can be file name, path or even a directory (where all files are included, except files beginning with ~ (tilde) or . (dot)). Including a directory is not recursive.

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

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

```
@version: 3.2
include "example.conf";
```

But the following is not:

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



Warning

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

5.3.2. Reusing configuration blocks

Starting with syslog-ng OSE 3.2, parts of a configuration file can be easily reused, you have to define the block (for example, a source) once, and reference it later. Any syslog-ng object can be a block. Use the following syntax to define a block:

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

For example, the elements of the *Source Configuration Library (SCL)* are configurations blocks. Blocks may be nested into each other, so for example an SCL may be built from other blocks.

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

- Include the file defining the block in the `syslog-ng.conf` file — or a file already included into `syslog-ng.conf`.



- Reference the name of the block in your configuration file. This will insert the block into your configuration. For example, to use a block called *myblock*, include the following line in your configuration:

```
myblock()
```

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



Example 5.4. Reusing configuration blocks

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

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

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

```
@version: 3.2
include "<correct/path>/myblocks.conf";
source s_myappsource myappsource()
...
log { source(s_myappsource); destination(...); }
```

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



Example 5.5. Defining blocks with multiple elements

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

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



Tip

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

If you want to reuse more than a single configuration object, for example, a logpath and the definitions of its sources and destinations, use the `include` feature to reuse the entire snippet. For details, see *Section 5.3.1, Including configuration files (p. 41)*.

5.3.2.1. 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 empty parentheses 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 5.6. Using arguments in blocks**

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

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



Chapter 6. Collecting log messages — sources and source drivers

6.1. How sources work

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

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

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



Example 6.1. A simple source statement

The following source statement receives messages on the TCP port 1999 of the interface having the 10.1.2.3 IP address.

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



Example 6.2. A source statement using two source drivers

The following source statement receives messages on the 1999 TCP port and the 1999 UDP port of the interface having the 10.1.2.3 IP address.

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



Example 6.3. Setting default priority and facility

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

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

Define a source only once. The same source can be used in several log paths. Duplicating sources causes syslog-ng to open the source (TCP/IP port, file, 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.

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

Platform	Method
Linux	A <code>SOCK_STREAM</code> unix socket named <code>/dev/log</code> ; some of the distributions switched over to using <code>SOCK_DGRAM</code> , though applications still work with either method.
BSD flavors	A <code>SOCK_DGRAM</code> unix socket named <code>/var/run/log</code> .
Solaris (2.5 or below)	An SVR4 style <code>STREAMS</code> device named <code>/dev/log</code> .



Platform	Method
Solaris (2.6 or above)	In addition to the <i>STREAMS</i> device used in earlier versions, 2.6 uses a new multithreaded IPC method called door. By default the door used by <i>syslogd</i> is <i>/etc/.syslog_door</i> .
HP-UX 11 or later	HP-UX uses a named pipe called <i>/dev/log</i> that is padded to 2048 bytes, for example <code>source s_hp-ux {pipe (" /dev/log" pad_size(2048) }.</code>
AIX 5.2 and 5.3	A <i>SOCK_STREAM</i> or <i>SOCK_DGRAM</i> unix socket called <i>/dev/log</i> .

Table 6.1. Communication methods used between the applications and *syslogd*

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



Example 6.4. Source statement on a Linux based operating system

The following source statement collects the following log messages:

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

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

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

Name	Description
<i>internal()</i>	Messages generated internally in <i>syslog-ng</i> .
<i>file()</i>	Opens the specified file and reads messages.
<i>pipe(), fifo</i>	Opens the specified named pipe and reads messages.
<i>pacct()</i>	Reads messages from the process accounting logs on Linux.
<i>program()</i>	Opens the specified application and reads messages from its standard output.
<i>sun-stream(), sun-streams()</i>	Opens the specified <i>STREAMS</i> device on Solaris systems and reads incoming messages.
<i>syslog()</i>	Listens for incoming messages using the new <i>IETF-standard syslog protocol</i> .
<i>system()</i>	Automatically detects which platform <i>syslog-ng</i> OSE is running on, and collects the native log messages of that platform.
<i>tcp(), tcp6()</i>	Listens on the specified TCP port for incoming messages using the <i>BSD-syslog protocol</i> over IPv4 and IPv6 networks, respectively.
<i>udp(), udp6()</i>	Listens on the specified UDP port for incoming messages using the <i>BSD-syslog protocol</i> over IPv4 and IPv6 networks, respectively.
<i>unix-dgram()</i>	Opens the specified unix socket in <i>SOCK_DGRAM</i> mode and listens for incoming messages.



Name	Description
<code>unix-stream()</code>	Opens the specified unix socket in <code>SOCK_STREAM</code> mode and listens for incoming messages.

Table 6.2. Source drivers available in `syslog-ng`

6.2. Collecting internal messages

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

```
internal()
```

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



Example 6.5. Using the `internal()` driver

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

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

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

6.3. Collecting messages from text files

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

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

The file driver has a single required parameter specifying the file to open. For the list of available optional parameters, see *Section 6.3.1, `file()` source options (p. 48)*.

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

**Example 6.6. Using the file() driver**

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

**Example 6.7. Tailing files**

The following source checks the `access.log` file every second for new messages.

```
source s_tail { file("/var/log/apache/access.log"
    follow_freq(1) flags(no-parse)); };
```

**Note**

If the message does not have a proper syslog header, syslog-ng treats messages received from files as sent by the `kern` facility. Use the `default-facility` and `default-priority` options in the source definition to assign a different facility if needed.

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

**Note**

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

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

6.3.1. file() source options

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

default-facility()

Type: facility string

Default: kern

Description: This parameter assigns a facility value to the messages received from the file source, if the message does not specify one.

default-priority()

Type: priority string

Default:

Description: This parameter assigns an emergency level to the messages received from the file source, if the message does not specify one.



file()

Type: filename with path

Default:

Description: The file to read messages from.

encoding()

Type: string

Default:

Description: Specifies the characterset (encoding, for example *UTF-8*) of messages using the legacy BSD-syslog protocol. To list the available character sets on a host, execute the `iconv -l` command.

flags()

Type: empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8

Default: empty set

Description: Specifies the log parsing options of the source.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *kernel*: The *kernel* flag makes the source default to the *LOG_KERN* | *LOG_CRIT* priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is \$PROGRAM instead of \$HOST. For example:

```
source s_dell { udp(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.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

- *no-parse*: The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- *store-legacy-msghdr*: If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.



- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.
- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

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

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global <i>log_fetch_limit()</i> option, which defaults to <i>10</i> .

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

log_iw_size()

Type:	number
Default:	100

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

log_msg_size()

Type:	number
Default:	Use the global <i>log_msg_size()</i> option, which defaults to <i>8192</i> .

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.



log_prefix() (DEPRECATED)

Type: string

Default:

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

optional()

Type: yes or no

Default:

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the *pipe()*, *unix-dgram*, and *unix-stream* drivers.

pad_size()

Type: number

Default: 0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where /dev/log is a named pipe and every write is padded to 2048 bytes. If *pad_size* was given and the incoming message does not fit into *pad_size*, syslog-ng will not read anymore from this pipe and displays the following error message:

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

program_override()

Type: string

Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

tags()

Type: string

Default:

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



time_zone()

Type: timezone in +/HH:MM format

Default:

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

6.4. Collecting messages from named pipes

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

The pipe driver has a single required parameter, specifying the filename of the pipe to open. For the list of available optional parameters, see *Section 6.4.1, pipe() source options (p. 52)*.

Declaration:

```
pipe(filename);
```



Note

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

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



Warning

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



Example 6.8. Using the pipe() driver

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

6.4.1. pipe() source options

The `pipe` driver has the following options:

flags()

Type: empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8

Default: empty set

Description: Specifies the log parsing options of the source.



- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *kernel*: The *kernel* flag makes the source default to the *LOG_KERN* | *LOG_CRIT* priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is \$PROGRAM instead of \$HOST. For example:

```
source s_dell { udp(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.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

- *no-parse*: The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- *store-legacy-msghdr*: If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.
- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.
- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

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



keep_timestamp()

Type: yes or no

Default: yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type: number

Default: The value specified by the global `log_fetch_limit()` option, which defaults to 10.

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

log_iw_size()

Type: number

Default: 100

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

log_msg_size()

Type: number

Default: Use the global `log_msg_size()` option, which defaults to 8192.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type: string

Default:

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

optional()

Type: yes or no

Default:

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

**pad_size()**

Type: number

Default: 0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where /dev/log is a named pipe and every write is padded to 2048 bytes. If *pad_size* was given and the incoming message does not fit into *pad_size*, syslog-ng will not read anymore from this pipe and displays the following error message:

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

pipe()

Type: filename with path

Default:

Description: The filename of the pipe to read messages from.

program_override()

Type: string

Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

tags()

Type: string

Default:

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

time_zone()

Type: timezone in +/HH:MM format

Default:

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



6.5. Collecting process accounting logs on Linux

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

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

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

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

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

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

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

```
@version: 3.2
@include "scl.conf"
source s_pacct { pacct(); };
...
log { source(s_pacct); destination(...); };
```



Note

The *pacct()* driver is actually an element of the syslog-ng *Source Configuration Library (SCL)*, a reusable configuration snippet tailored to handle process accounting logs. For details on using or writing SCLs, see [???](#).

6.5.1. pacct() options

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

file

Type:	filename with path
Default:	<code>/var/log/account/pacct</code>



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

follow_freq()

Type: number

Default: 1

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

6.6. Receiving messages from external applications

The program driver starts an external application and reads messages from the standard output (stdout) of the application. It is mainly useful to receive log messages from daemons that accept incoming messages and convert them to log messages.

The program driver has a single required parameter, specifying the name of the application to start.

Declaration:

```
program(filename);
```



Example 6.9. Using the program() driver

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



Note

The program is restarted automatically if it exits.

6.6.1. program() source options

The *program* driver has the following options:

flags()

Type: empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8

Default: empty set

Description: Specifies the log parsing options of the source.

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



- *kernel*: The *kernel* flag makes the source default to the *LOG_KERN* | *LOG_CRIT* priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is \$PROGRAM instead of \$HOST. For example:

```
source s_dell { udp(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.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

- *no-parse*: The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- *store-legacy-msghdr*: If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.
- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.
- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type: number

Default: 1

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

keep_timestamp()

Type: yes or no

Default: yes



Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type: number

Default: The value specified by the global `log_fetch_limit()` option, which defaults to 10.

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

log_iw_size()

Type: number

Default: 100

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

log_msg_size()

Type: number

Default: Use the global `log_msg_size()` option, which defaults to 8192.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type: string

Default:

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

optional()

Type: yes or no

Default:

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

pad_size()

Type: number

Default: 0



Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-*ng* OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where */dev/log* is a named pipe and every write is padded to 2048 bytes. If *pad_size* was given and the incoming message does not fit into *pad_size*, syslog-*ng* will not read anymore from this pipe and displays the following error message:

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

program

Type: filename with path
Default:

Description: The name of the application to start and read messages from.

program_override()

Type: string
Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override ("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

tags()

Type: string
Default:

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

time_zone()

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

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

6.7. Collecting messages on Sun Solaris

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

**Note**

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

The *sun-streams()* driver has a single required argument specifying the *STREAMS* device to open, and the *door()* option. For the list of available optional parameters, see *Section 6.7.1, sun-streams() source options (p. 61)*.

Declaration:

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

**Example 6.10. Using the sun-streams() driver**

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

6.7.1. sun-streams() source options

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

door()

Type: string

Default: none

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

flags()

Type: empty-lines, kernel, no-multi-line, no-parse, store-legacy-msghdr, syslog-protocol, validate-utf8

Default: empty set

Description: Specifies the log parsing options of the source.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *kernel*: The *kernel* flag makes the source default to the *LOG_KERN* | *LOG_CRIT* priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is *\$PROGRAM* instead of *\$HOST*. For example:

```
source s_dell { udp(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.



By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the `no-parse` flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

- `no-parse`: The `no-parse` flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- `store-legacy-msghdr`: If the `store-legacy-msghdr` flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before `msg` in the following message: `Jan 22 10:06:11 host program:msg`). Note that `store-legacy-msghdr` should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.
- `syslog-protocol`: The `syslog-protocol` flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the `syslog` driver.
- `validate-utf8`: The `validate-utf8` flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

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

keep_timestamp()

Type:	yes or no
Default:	yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type:	number
Default:	The value specified by the global <code>log_fetch_limit()</code> option, which defaults to 10.



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

`log_iw_size()`

Type: number

Default: 100

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

`log_msg_size()`

Type: number

Default: Use the global `log_msg_size()` option, which defaults to 8192.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

`log_prefix() (DEPRECATED)`

Type: string

Default:

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

`optional()`

Type: yes or no

Default:

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the `pipe()`, `unix-dgram`, and `unix-stream` drivers.

`pad_size()`

Type: number

Default: 0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in `pad_size()`. Mostly used on HP-UX where `/dev/log` is a named pipe and every write is padded to 2048 bytes. If `pad_size` was given and the incoming message does not fit into `pad_size`, syslog-ng will not read anymore from this pipe and displays the following error message:

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



program_override()

Type: string

Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the `program_override ("kernel")` option in the source containing `/proc/kmsg`. NOTE: This option replaces the deprecated `log_prefix()` option.

tags()

Type: string

Default:

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

time_zone()

Type: timezone in +/HH:MM format

Default:

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

6.8. Collecting messages using the IETF syslog protocol

The `syslog()` driver enables to receive messages from the network using the standard syslog protocols and message formats. The `syslog()` driver can handle and automatically recognize messages sent using both the BSD-syslog (described in RFC 3164, see *Section 2.9.1, BSD-syslog or legacy-syslog messages (p. 10)*) and the newer IETF-syslog (described in RFC 5424-28, see *Section 2.9.2, IETF-syslog messages (p. 12)*) protocols. UDP, TCP, and TLS-encrypted TCP can all be used to transport the messages.



Note

In syslog-ng OSE versions 3.1 and earlier, the `syslog()` driver could handle only IETF-syslog (RFC 5424-28) messages.

For the list of available optional parameters, see *Section 6.8.1, syslog() source options (p. 65)*.

Declaration:

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



Example 6.11. Using the `syslog()` driver

TCP source listening on the localhost on port 1999.

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

UDP source with defaults.



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

Encrypted source where the client is also authenticated. For details on the encryption settings, see *Section 10.4, “TLS options (p. 151)*.

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

As a general rule, increase the *so_recvbuf()* 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_recvbuf()* at least to *2 097 152* bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

6.8.1. syslog() source options

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

flags()

Type:	<i>empty-lines</i> , <i>kernel</i> , <i>no-multi-line</i> , <i>no-parse</i> , <i>store-legacy-msghdr</i> , <i>syslog-protocol</i> , <i>validate-utf8</i>
Default:	<i>empty set</i>

Description: Specifies the log parsing options of the source.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *kernel*: The *kernel* flag makes the source default to the *LOG_KERN | LOG_CRIT* priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is \$PROGRAM instead of \$HOST. For example:

```
source s_dell { udp(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.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.



- *no-parse*: The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- *store-legacy-msghdr*: If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.
- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.
- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

follow_freq()

Type:	number
Default:	1

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

host_override()

Type:	string
Default:	

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

ip() or localip()

Type:	string
Default:	0.0.0.0

Description: The IP address to bind to. Note that this is not the address where messages are accepted from.

ip_tos()

Type:	number
Default:	0

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

**ip_ttl()**

Type: number

Default: 0

Description: Specifies the Time-To-Live value of outgoing packets.**keep-alive()**

Type: yes or no

Default: yes

Description: Specifies whether connections to sources should be closed when syslog-ng is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the syslog-ng connections, client-side (destination) connections are always reopened after receiving a HUP signal unless the *keep-alive* option is enabled for the destination.**keep_hostname()**

Type: yes or no

Default: no

Description: Enable or disable hostname rewriting.

- If enabled (*keep_hostname (yes)*), syslog-ng OSE assumes that the incoming log message was sent by the host specified in the *HOST* field of the message.
- If disabled (*keep_hostname (no)*), syslog-ng OSE rewrites the *HOST* field of the message, either to the IP address (if the *use_dns ()* parameter is set to *no*), or to the hostname (if the *use_dns ()* parameter is set to *yes* and the IP address can be resolved to a hostname) of the host sending the message to syslog-ng OSE. For details on using name resolution in syslog-ng OSE, see *Section 16.4, Using name resolution in syslog-ng (p. 197)*.

**Note**

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

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

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

**Note**

When relaying messages, enable this option on the syslog-ng OSE server and also on every relay, otherwise syslog-ng OSE will treat incoming messages as if they were sent by the last relay.

**keep_timestamp()**

Type: yes or no

Default: yes

Description: Specifies whether syslog-*ng* should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type: number

Default: The value specified by the global `log_fetch_limit()` option, which defaults to 10.

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

log_iw_size()

Type: number

Default: 100

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

log_msg_size()

Type: number

Default: Use the global `log_msg_size()` option, which defaults to 8192.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type: string

Default:

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

max-connections()

Type: number

Default: 10

Description: Specifies the maximum number of simultaneous connections.

**pad_size()**

Type: number

Default: 0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where /dev/log is a named pipe and every write is padded to 2048 bytes. If *pad_size* was given and the incoming message does not fit into *pad_size*, syslog-ng will not read anymore from this pipe and displays the following error message:

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

port() or localport()

Type: number

Default: 514

Description: The port number to bind to.

program_override()

Type: string

Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.

so_broadcast()

Type: yes or no

Default: no

Description: This option controls the *SO_BROADCAST* socket option required to make syslog-ng send messages to a broadcast address. 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 syslog-ng OSE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so_recvbuf()` option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the `so_recvbuf()` 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_recvbuf()` at least to `2 097 152` bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

so_sndbuf()

Type: number

Default: 0

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

tags()

Type: string

Default:

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

tcp-keep-alive()

Type: yes or no

Default: no

Description: This is an obsolete alias of the `so_keepalive()` option.

time_zone()

Type: timezone in +/HH:MM format

Default:

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

transport()

Type: udp, tcp, or tls

Default: tcp

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

**Warning**

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

As a general rule, increase the `so_recvbuf()` 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_recvbuf()` at least to `2 097 152` bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

tls()

Type:	tls options
Default:	n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations and can only be used with the `tcp` transport protocols. For details, see *Section 10.4, “TLS options” (p. 151)*.

use_dns()

Type:	yes, no, persist_only
Default:	yes

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from file (for example from `/etc/hosts`). syslog-ng blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all hosts which may get to syslog-ng are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

use_fqdn()

Type:	yes or no
Default:	no

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

6.9. Collecting the system-specific log messages of a platform

Starting with version 3.2, syslog-ng OSE can automatically collect the system-specific log messages of the host on a number of platforms using the `system()` driver. If the `system()` driver is included in the syslog-ng OSE configuration file, syslog-ng OSE executes the `/opt/syslog-ng/scl/system/generate-system-source.sh` script during startup, and adds the following sources to the syslog-ng OSE configuration.

The `system()` driver is also used in the default configuration file of syslog-ng OSE. For details on the default configuration file, see *Example 4.1, “The default configuration file of syslog-ng OSE” (p. 33)*.

**Warning**

If syslog-ng OSE does not recognize the platform it is installed on, it does not add any sources.

Platform	Message source
Linux	<code>unix-dgram("/dev/log");</code> <code>file("/proc/kmsg" program-override("kernel") flags(kernel));</code>
Solaris 8	<code>sun-streams("/dev/log");</code>
Solaris 9	<code>sun-streams("/dev/log" door("/etc/.syslog_door"));</code>
Solaris 10	<code>sun-streams("/dev/log" door("/var/run/syslog_door"));</code>
FreeBSD	<code>unix-dgram("/var/run/log");</code> <code>unix-dgram("/var/run/logpriv" perm(0600));</code> <code>file("/dev/klog" follow-freq(0) program-override("kernel") flags(no-parse));</code>
HP-UX	<code>pipe("/dev/log" pad_size(2048));</code>
AIX and Tru64	<code>unix-dgram("/dev/log");</code>

Table 6.3. Sources automatically added by *syslog-ng Open Source Edition*

**Note**

The `system()` driver is actually an element of the *syslog-ng Source Configuration Library (SCL)*, a reusable configuration snippet tailored to handle process accounting logs. For details on using or writing SCLs, see [???](#).

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

**Note**

Starting with *syslog-ng OSE* version 3.2, the `syslog()` source driver can handle both BSD-syslog (RFC 3164) and IETF-syslog (RFC 5424-28) messages. It is recommended to use the `syslog()` driver for both types of traffic, as the `tcp()`, `tcp6()`, `udp()`, `udp6()` source drivers will be deprecated in later versions.

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.



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

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

The *tcp()* and *udp()* drivers do not have any required parameters. By default they bind to the *0.0.0.0:514* address, which means that syslog-*ng* will listen on all available interfaces, port 514. To limit accepted connections to only one interface, use the *localip()* parameter as described below. For the list of available optional parameters, see *Section 6.10.1, tcp(), tcp6(), udp() and udp6() source options (p. 74)*.

Declaration:

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



Note

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

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

The syslog-*ng* application supports TLS (Transport Layer Security, also known as SSL) for the *tcp()* and *tcp6()* drivers. For details, see the TLS-specific options below and *Section 10.2, Encrypting log messages with TLS (p. 146)*. For the list of available optional parameters, see *Section 6.10.1, tcp(), tcp6(), udp() and udp6() source options (p. 74)*.



Tip

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



Example 6.12. Using the *udp()* and *tcp()* drivers

A simple *udp()* source with default settings.

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

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

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

A TCP source listening on a TLS-encrypted channel.

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



A TCP source listening for messages using the IETF-syslog message format. Note that for transferring IETF-syslog messages, generally you are recommended to use the `syslog()` driver on both the client and the server, as it uses both the IETF-syslog message format and the protocol. For details, see *Section 6.8, “Collecting messages using the IETF syslog protocol (p. 64)*.

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

6.10.1. tcp(), tcp6(), udp() and udp6() source options

The `tcp()`, `tcp6()`, `udp()`, `udp6()` drivers can receive messages conforming to RFC3164 from the network using the TCP and UDP networking protocols.

The following options are valid for `tcp()`, `tcp6()`, `udp()`, and `udp6()` drivers:



Warning

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

As a general rule, increase the `so_recvbuf()` 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_recvbuf()` at least to `2 097 152` bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

encoding()

Type: string
Default:

Description: Specifies the characterset (encoding, for example *UTF-8*) of messages using the legacy BSD-syslog protocol. To list the available character sets on a host, execute the `iconv -l` command.

flags()

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

Description: Specifies the log parsing options of the source.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *kernel*: The *kernel* flag makes the source default to the `LOG_KERN | LOG_CRIT` priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is `$PROGRAM` instead of `$HOST`. For example:

```
source s_dell { udp(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.

By default, syslog-ng parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-ng will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

- *no-parse*: The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- *store-legacy-msghdr*: If the *store-legacy-msghdr* flag is enabled, syslog-ng stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-ng automatically corrects minor header errors, for example, adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-ng Agent for Windows clients that use the Snare-compatible mode.
- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.
- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-ng automatically adds the BOM character to the message.

host_override()

Type: string

Default:

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

ip() or localip()

Type: string

Default: 0.0.0.0

Description: The IP address to bind to. Note that this is not the address where messages are accepted from.

keep-alive()

Type: yes or no

Default: yes

Description: Specifies whether connections to sources should be closed when syslog-ng is forced to reload its configuration (upon the receipt of a SIGHUP signal). Note that this applies to the server (source) side of the syslog-ng connections, client-side (destination) connections are always reopened after receiving a HUP signal unless the *keep-alive* option is enabled for the destination.



keep_hostname()

Type: yes or no

Default: no

Description: Enable or disable hostname rewriting.

- If enabled (*keep_hostname (yes)*), syslog-ng OSE assumes that the incoming log message was sent by the host specified in the *HOST* field of the message.
- If disabled (*keep_hostname (no)*), syslog-ng OSE rewrites the *HOST* field of the message, either to the IP address (if the *use_dns ()* parameter is set to *no*), or to the hostname (if the *use_dns ()* parameter is set to *yes* and the IP address can be resolved to a hostname) of the host sending the message to syslog-ng OSE. For details on using name resolution in syslog-ng OSE, see *Section 16.4, Using name resolution in syslog-ng* (p. 197).

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

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

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

**Note**

When relaying messages, enable this option on the syslog-ng OSE server and also on every relay, otherwise syslog-ng OSE will treat incoming messages as if they were sent by the last relay.

keep_timestamp()

Type: yes or no

Default: yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type: number

Default: The value specified by the global *log_fetch_limit ()* option, which defaults to 10.**Description:** The maximum number of messages fetched from a source during a single poll loop. The destination queues might fill up before flow-control could stop reading if *log_fetch_limit ()* is too high.



log_iw_size()

Type: number

Default: 100

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

log_msg_size()

Type: number

Default: Use the global *log_msg_size()* option, which defaults to 8192.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type: string

Default:

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

max-connections()

Type: number

Default: 10

Description: Specifies the maximum number of simultaneous connections.

pad_size()

Type: number

Default: 0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where /dev/log is a named pipe and every write is padded to 2048 bytes. If *pad_size* was given and the incoming message does not fit into *pad_size*, syslog-ng will not read anymore from this pipe and displays the following error message:

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

port() or localport()

Type: number

Default: 514



Description: The port number to bind to.

program_override()

Type: string

Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the `program_override("kernel")` option in the source containing `/proc/kmsg`. NOTE: This option replaces the deprecated `log_prefix()` option.

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

As a general rule, increase the `so_rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so_rcvbuf()` at least to `2 097 152` bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

tcp-keep-alive()

Type: yes or no

Default: no

Description: This is an obsolete alias of the `so_keepalive()` option.

tags()

Type: string

Default:



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

time_zone()

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

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

tls()

Type:	tls options
Default:	n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations and can only be used with the `tcp` transport protocols. For details, see *Section 10.4, “TLS options (p. 151)*.

use_dns()

Type:	yes, no, persist_only
Default:	yes

Description: Enable or disable DNS usage. The `persist_only` option attempts to resolve hostnames locally from file (for example from `/etc/hosts`). syslog-ng blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all hosts which may get to syslog-ng are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

use_fqdn()

Type:	yes or no
Default:	no

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

6.11. Collecting messages from UNIX domain sockets

The `unix-stream()` and `unix-dgram()` drivers open an `AF_UNIX` socket and start listening on it for messages. The `unix-stream()` driver is primarily used on Linux and uses `SOCK_STREAM` semantics (connection oriented, no messages are lost); while `unix-dgram()` is used on BSDs and uses `SOCK_DGRAM` semantics: this may result in lost local messages if the system is overloaded.

To avoid denial of service attacks when using connection-oriented protocols, the number of simultaneously accepted connections should be limited. This can be achieved using the `max-connections()` parameter. The default value of this parameter is quite strict, you might have to increase it on a busy system.



Both unix-stream and unix-dgram have a single required argument that specifies the filename of the socket to create. For the list of available optional parameters, see *Section 6.11.1, unix-stream() and unix-dgram() source options (p. 80)*

Declaration:

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



Note

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



Example 6.13. Using the unix-stream() and unix-dgram() drivers

```
source s_stream { unix-stream("/dev/log" max-connections(10)); };
source s_dgram { unix-dgram("/var/run/log"); };
```

6.11.1. 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 divers:

encoding()

Type:	string
Default:	

Description: Specifies the charset (encoding, for example *UTF-8*) of messages using the legacy BSD-syslog protocol. To list the available character sets on a host, execute the *iconv -l* command.

flags()

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

Description: Specifies the log parsing options of the source.

- *empty-lines*: Use the *empty-lines* flag to keep the empty lines of the messages. By default, syslog-ng OSE removes empty lines automatically.
- *kernel*: The *kernel* flag makes the source default to the *LOG_KERN | LOG_CRIT* priority if not specified otherwise.
- *no-hostname*: Enable the *no-hostname* flag if the log message does not include the hostname of the sender host. That way syslog-ng OSE assumes that the first part of the message header is \$PROGRAM instead of \$HOST. For example:

```
source s_dell { udp(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.

By default, syslog-*ng* parses incoming messages as syslog messages. If a source does not send properly formatted messages, use the *no-parse* flag to disable message parsing for the source. As a result, syslog-*ng* will generate a new syslog header and put the entire incoming message into the MSG part of the syslog message.

- *no-parse*: The *no-parse* flag completely disables syslog message parsing and processes the complete line as the message part of a syslog message. Other information (timestamp, host, and so on) is added automatically. This flag is useful for parsing files not complying to the syslog format.
- *store-legacy-msghdr*: If the *store-legacy-msghdr* flag is enabled, syslog-*ng* stores the original incoming header of the log message. This is useful if the original format of a non-syslog-compliant message must be retained (syslog-*ng* automatically corrects minor header errors, for example, adds a whitespace before *msg* in the following message: *Jan 22 10:06:11 host program:msg*). Note that *store-legacy-msghdr* should be enabled when receiving messages from syslog-*ng* Agent for Windows clients that use the Snare-compatible mode.
- *syslog-protocol*: The *syslog-protocol* flag specifies that incoming messages are expected to be formatted according to the new IETF syslog protocol standard. Note that this flag is not needed for the *syslog* driver.
- *validate-utf8*: The *validate-utf8* flag enables encoding-verification for messages formatted according to the new IETF syslog standard (for details, see *Section 2.9.2, IETF-syslog messages (p. 12)*). If the BOM character is missing, but the message is otherwise UTF-8 compliant, syslog-*ng* automatically adds the BOM character to the message.

group()

Type:	string
Default:	root

Description: Set the gid of the socket.

host_override()

Type:	string
Default:	

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

keep-alive()

Type:	yes or no
Default:	yes

Description: Selects whether to keep connections open when syslog-*ng* is restarted; cannot be used with *unix-dgram()*.



keep_timestamp()

Type: yes or no

Default: yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

log_fetch_limit()

Type: number

Default: The value specified by the global `log_fetch_limit()` option, which defaults to 10.

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

log_iw_size()

Type: number

Default: 100

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

log_msg_size()

Type: number

Default: Use the global `log_msg_size()` option, which defaults to 8192.

Description: Specifies the maximum length of incoming log messages. Uses the value of the *global option* if not specified.

log_prefix() (DEPRECATED)

Type: string

Default:

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

max-connections()

Type: number

Default: 256

Description: Limits the number of simultaneously open connections. Cannot be used with `unix-dgram()`.



optional()

Type: yes or no

Default:

Description: Instruct syslog-ng to ignore the error if a specific source cannot be initialized. No other attempts to initialize the source will be made until the configuration is reloaded. This option currently applies to the *pipe()*, *unix-dgram*, and *unix-stream* drivers.

owner()

Type: string

Default: root

Description: Set the uid of the socket.

pad_size()

Type: number

Default: 0

Description: Specifies input padding. Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes). The syslog-ng OSE application will pad reads from the associated device to the number of bytes set in *pad_size()*. Mostly used on HP-UX where /dev/log is a named pipe and every write is padded to 2048 bytes. If *pad_size* was given and the incoming message does not fit into *pad_size*, syslog-ng will not read anymore from this pipe and displays the following error message:

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

perm()

Type: number

Default: 0666

Description: Set the permission mask. For octal numbers prefix the number with '0', for example: use 0755 for rwxr-xr-x.

program_override()

Type: string

Default:

Description: Replaces the \$PROGRAM part of the message with the parameter string. For example, to mark every message coming from the kernel, include the *program_override ("kernel")* option in the source containing */proc/kmsg*. NOTE: This option replaces the deprecated *log_prefix()* option.



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

As a general rule, increase the `so_rcvbuf()` so that the buffer size in kilobytes is higher than the rate of incoming messages per second. For example, to receive 2000 messages per second, set the `so_rcvbuf()` at least to 2 097 152 bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

tags()

Type: string

Default:

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

time_zone()

Type: timezone in +/HH:MM format

Default:

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



Chapter 7. Sending and storing log messages — destinations and destination drivers

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



Tip

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

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

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



Example 7.1. A simple destination statement

The following destination statement sends messages to the TCP port 1999 of the 10.1.2.3 host.

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

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

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

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

Name	Description
<i>file()</i>	Writes messages to the specified file.
<i>fifo(), pipe()</i>	Writes messages to the specified named pipe.
<i>program()</i>	Forks and launches the specified program, and sends messages to its standard input.
<i>sql()</i>	Sends messages into an SQL database. In addition to the standard syslog- <i>ng</i> packages, the <i>sql()</i> destination requires database-specific packages to be installed. Refer to the section appropriate for your platform in <i>Chapter 3, Installing syslog-<i>ng</i> (p. 16)</i> .
<i>syslog()</i>	Sends messages to the specified remote host using the <i>IETF-syslog protocol</i> . The IETF standard supports message transport using the UDP, TCP, and TLS networking protocols.
<i>tcp() and tcp6()</i>	Sends messages to the specified TCP port of a remote host using the <i>BSD-syslog protocol</i> over IPv4 and IPv6, respectively.
<i>udp() and udp6()</i>	Sends messages to the specified UDP port of a remote host using the <i>BSD-syslog protocol</i> over IPv4 and IPv6, respectively.
<i>unix-dgram()</i>	Sends messages to the specified unix socket in <i>SOCK_DGRAM</i> style (BSD).



Name	Description
<code>unix-stream()</code>	Sends messages to the specified unix socket in <code>SOCK_STREAM</code> style (Linux).
<code>usertty()</code>	Sends messages to the terminal of the specified user, if the user is logged in.

Table 7.1. Destination drivers available in `syslog-ng`

7.1. Storing messages in plain-text files

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

The destination filename may include macros which get expanded when the message is written, thus a simple `file()` driver may create several files: for example, `syslog-ng` OSE can store the messages of client hosts in a separate file for each host. For more information on available macros see *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*.

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

The `file()` has a single required parameter that specifies the filename that stores the log messages. For the list of available optional parameters, see *Section 7.1.1, file() destination options (p. 87)*.

Declaration:

```
file(filename options());
```



Example 7.2. Using the file() driver

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



Example 7.3. Using the file() driver with macros in the file name and a template for the message

```
destination d_file {
    file("/var/log/$YEAR.$MONTH.$DAY/messages"
        template("$HOUR:$MIN:$SEC $TZ $HOST [$LEVEL] $MSG $MSG\n")
        template_escape(no));
};
```



Note

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



Warning

Since the state of each created file must be tracked by `syslog-ng`, it consumes some memory for each file. If no new messages are written to a file within 60 seconds (controlled by the `time_reap()` global option), it is closed, and its state is freed.

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

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



7.1.1. 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, syslog-ng might not be able to open the required number of files. This might happen for example when using the `$HOST` macro in the filename while receiving messages from a large number of hosts. To overcome this problem, adjust the `--fd-limit` command-line parameter of syslog-ng or the global ulimit parameter of your host. For setting the `--fd-limit` command-line parameter of syslog-ng see the `syslog(8)` (p. 201) manual page. For setting the `ulimit` parameter of the host, see the documentation of your operating system.

create_dirs()

Type: yes or no

Default: no

Description: Enable creating non-existing directories.

dir_group()

Type: string

Default: root

Description: The group of directories created by syslog-ng.

dir_owner()

Type: string

Default: root

Description: The owner of directories created by syslog-ng.

dir_perm()

Type: number

Default: Use the global settings

Description: The permission mask of directories created by syslog-ng. Log directories are only created if a file after macro expansion refers to a non-existing directory, and directory creation is enabled (see also the `create_dirs()` option). For octal numbers prefix the number with `0`, for example use `0755` for `rwxr-xr-x`.

To preserve the original properties of an existing directory, use the option without specifying an attribute: `dir_perm()`. Note that when creating a new directory without specifying attributes for `dir_perm()`, the default permission of the directories is masked with the umask of the parent process (typically `0022`).

**flags()**

Type:	no_multi_line, syslog-protocol
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. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-*ng* waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-*ng* waits for lines to accumulate in its output buffer. For details, see the *flush_lines* option.

frac_digits()

Type:	number
Default:	0

Description: The syslog-*ng* application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-*ng* can add the fractions to non-ISO8601 timestamps as well.

fsync()

Type:	yes or no
Default:	no



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

group()

Type:	string
Default:	root

Description: Set the group of the created file to the one specified.

local_time_zone()

Type:	name of the timezone or the timezone offset
Default:	The local timezone.

Description: Sets the timezone used when expanding filename and tablename templates. The timezone can be specified as using the name of the (for example `time_zone("Europe/Budapest")`), or as the timezone offset (for example `+01:00`). The valid timezone names are listed under the `/usr/share/zoneinfo` directory.

log_fifo_size()

Type:	number
Default:	Use global setting.

Description: The number of entries in the output buffer (output fifo).

overwrite_if_older()

Type:	number
Default:	0

Description: If set to a value higher than 0, syslog-ng checks when the file was last modified before starting to write into the file. If the file is older than the specified amount of time (in seconds), then syslog-ng removes the existing file and opens a new file with the same name. In combination with for example the `$WEEKDAY` macro, this can be used for simple log rotation, in case not all history has to be kept. (Note that in this weekly log rotation example if its Monday 00:01, then the file from last Monday is not seven days old, because it was probably last modified shortly before 23:59 last Monday, so it is actually not even six days old. So in this case, set the `overwrite_if_older()` parameter to a-bit-less-than-six-days, for example, to 518000 seconds.)

owner()

Type:	string
Default:	root

Description: Set the owner of the created file to the one specified.



pad_size()

Type: number

Default: 0

Description: If set, syslog-ng OSE will pad output messages to the specified size (in bytes). Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes).

**Warning**

Hazard of data loss! If the size of the incoming message is larger than the previously set pad_size value, syslog-ng will truncate the message to the specified size. Therefore, all message content above that size will be lost.

perm()

Type: number

Default: 0600

Description: The permission mask of the file if it is created by syslog-ng. For octal numbers prefix the number with `0`, for example use `0755` for `rwxr-xr-x`.

suppress()

Type: seconds

Default: 0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type: string

Default: A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like `syslogd` or `syslog-ng` itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type: yes or no

Default: no



Description: Turns on escaping for the ', ", and backspace characters in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type: number

Default: 0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying 0 or a lower value sets the output limit to unlimited.

time_zone()

Type: timezone in +/HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

time_reap()

Accepted values: number

Default: Use global settings

Description: The time to wait in seconds before an idle destination file is closed. Note that only destination files having macros in their filenames are closed automatically.

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 *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

7.2. Sending messages to named pipes

The `pipe()` driver sends messages to a named pipe like `/dev/xconsole`.

The pipe driver has a single required parameter, specifying the filename of the pipe to open. The filename can include macros. For the list of available optional parameters, see *Section 7.2.1, pipe() destination options (p. 92)*.

Declaration:

```
pipe(filename);
```

**Warning**

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

**Example 7.4. Using the pipe() driver**

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

7.2.1. pipe() destination options

This driver sends messages to a named pipe like `/dev/xconsole`.

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

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.
- *syslog-protocol*: The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the `flush_timeout` option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the `flush_lines` option.

**frac_digits()**

Type: number

Default: 0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

group()

Type: string

Default: root

Description: Set the group of the pipe to the one specified.

log_fifo_size()

Type: number

Default: Use global setting.

Description: The number of entries in the output buffer (output fifo).

owner()

Type: string

Default: root

Description: Set the owner of the pipe to the one specified.

pad_size()

Type: number

Default: 0

Description: If set, syslog-ng OSE will pad output messages to the specified size (in bytes). Some operating systems (such as HP-UX) pad all messages to block boundary. This option can be used to specify the block size. (HP-UX uses 2048 bytes).

**Warning**

Hazard of data loss! If the size of the incoming message is larger than the previously set pad_size value, syslog-ng will truncate the message to the specified size. Therefore, all message content above that size will be lost.

**perm()**

Type: number

Default: 0600

Description: The permission mask of the pipe. For octal numbers prefix the number with '0', for example: use 0755 for rwxr-xr-x.

suppress()

Type: seconds

Default: 0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type: string

Default: A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *syslogd* or *syslog-**ng*** itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type: yes or no

Default: no

Description: Turns on escaping for the ',', ", and backspace characters in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type: number

Default: 0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying 0 or a lower value sets the output limit to unlimited.



time_zone()

Type: timezone in +/-HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type: rfc3164, bsd, rfc3339, iso

Default: rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. For details, see *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

7.3. Sending messages to external applications

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

The `program()` driver has a single required parameter, specifying a program name to start. The program is executed with the help of the current shell, so the command may include both file patterns and I/O redirections. For the list of available optional parameters, see *Section 7.3.1, program() destination options (p. 96)*.

Declaration:

```
program(command_to_run);
```



Note

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



Warning

The syslog-ng OSE application must be able to start and restart the external program, and have the necessary permissions to do so. For example, if your host is running AppArmor, you might have to modify your AppArmor configuration to enable syslog-ng OSE to execute external applications.

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



Example 7.5. Using the program() destination driver

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



7.3.1. 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:

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. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-*ng* waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type:	time in milliseconds
Default:	Use global setting.

Description: Specifies the time syslog-*ng* waits for lines to accumulate in its output buffer. For details, see the *flush_lines* option.

frac_digits()

Type:	number
Default:	0

Description: The syslog-*ng* application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-*ng* can add the fractions to non-ISO8601 timestamps as well.

**log_fifo_size()**

Type: number

Default: Use global setting.

Description: The number of entries in the output buffer (output fifo).

suppress()

Type: seconds

Default: 0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type: string

Default: A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *syslogd* or *syslog-ng* itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type: yes or no

Default: no

Description: Turns on escaping for the ', ", and backspace characters in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type: number

Default: 0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying 0 or a lower value sets the output limit to unlimited.



time_zone()

Type: timezone in +/-HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type: rfc3164, bsd, rfc3339, iso

Default: rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. For details, see *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

7.4. Storing messages in an SQL database

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

Declaration:

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



Warning

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

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



Note

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

The `sql()` driver is currently not available for every platform that is supported by syslog-ng. For a list of platforms that support the `sql()` driver, [visit this link](#).

The `table` and `value` parameters can include macros to create tables and columns dynamically (for details, see *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*).



Warning

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

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

**Example 7.6. Using the sql() driver**

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

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

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

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

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

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

- The hostname of the database server is set in the *tnsnames.ora* file, not in the *host* parameter of the *sql()* destination.
Make sure to set the Oracle-related environment variables properly, so syslog-ng and the Oracle client will find the file. The following variables must be set: *ORACLE_BASE*, *ORACLE_HOME*, and *ORACLE_SID*. For details, see the documentation of the Oracle Instant Client.
- As certain database versions limit the maximum length of table names, macros in the table names should be used with care.
- In the current version of syslog-ng PE, the types of database columns must be explicitly set for the Oracle destination. The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng, therefore it is usually recommended to use the *varchar2* or *clob* column type. (The maximum length of the messages can be set using the *log_msg_size()* option.) For details, see the following example.

**Example 7.7. Using the sql() driver with an Oracle database**

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

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



```
values("$R_DATE", "$HOST", "$PROGRAM", "$PID", "$MSGONLY")
indexes("datetime", "host", "program", "pid", "message"));
};
```

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

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

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

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

- The date format used by the MSSQL database must be explicitly set in the `/etc/locales.conf` file of the syslog-ng server. For details, see the following example.
- As certain database versions limit the maximum length of table names, macros in the table names should be used with care.
- In the current version of syslog-ng PE, the types of database columns must be explicitly set for the MSSQL destination. The column used to store the text part of the syslog messages should be able to store messages as long as the longest message permitted by syslog-ng. The `varchar` column type can store maximum 4096 bytes-long messages. The maximum length of the messages can be set using the `log_msg_size()` option. For details, see the following example.
- Remote access for SQL users must be explicitly enabled on the Microsoft Windows host running the Microsoft SQL Server. For details, see *Procedure 3.6, Configuring Microsoft SQL Server to accept logs from syslog-ng (p. 26)*.



Example 7.8. Using the sql() driver with an MSSQL database

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

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

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

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



7.4.3. The way syslog-ng interacts with the database

Used SQL operations by syslog-ng.

Create table:

- If the given table does not exist, syslog-ng tries to create it with the given column types.
- The syslog-ng OSE application automatically creates the required tables and columns, if the user account used to connect to the database has the required privileges.
- If syslog-ng cannot create or alter a table, it tries to do it again when reach the next time_reopen.

Alter table:

- If the table structure is different from given structure in an existing table, syslog-ng tries to add columns in this table but never will delete or modify an existing column.
- If syslog-ng OSE cannot create or alter a table, it tries to do it again when reach the next time_reopen.
- The syslog-ng OSE application requires read and write access to the SQL table, otherwise it cannot verify that the destination table exists.

Insert table:

- Insert new records in a table.
- Inserting the records into the database is performed by a separate thread.
- The syslog-ng OSE application automatically performs the escaping required to insert the messages into the database.
- If insert returns with error, syslog-ng tries to insert the message +two times by default, then drops it. Retrying time is the value of time_reopen().

Encoding.

The syslog-ng OSE application uses UTF-8 by default when writes logs into database.

Start/stop and reload.

Start:

- The syslog-ng OSE application will connect to database automatically after starting regardless existing incoming messages.

Stop:

- The syslog-ng OSE application will close the connection to database before shutting down.

Possibility of losing logs:

- The syslog-ng OSE application cannot lose logs during shutting down if disk buffer was given and it is not full yet.
- The syslog-ng OSE application cannot lose logs during shutting down if disk buffer was not given.

Reload:

- The syslog-ng OSE application will close the connection to database if it received SIGHUP signal (reload).



- It will reconnect to the database when it tries to send a new message to this database again.

Macros:

The value of \$SEQNUM macro will be overrided by sql driver regardless of local or relayed incoming message.

It will be grown continuously.

7.4.3.1. 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`.

7.4.3.2. 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 syslog-`ng` server. `[default] date = "%Y-%m-%d %H:%M:%S"`.

7.4.4. sql() destination options

This driver sends messages into an SQL database. The `sql()` destination has the following options:

columns()

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

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

database()

Type:	string
Default:	logs

Description: Name of the database that stores the logs. Macros cannot be used in database name.

flags()

Type:	list of flags
Default:	empty string

Description: Flags related to the `sql()` destination.



- *dont-create-tables*: Enable this flag to prevent syslog-ng OSE from creating non-existing database tables automatically. The syslog-ng OSE application typically has to create tables if you use macros in the table names. Available in syslog-ng OSE version 3.2 and later.
- *explicit-commits*: By default, syslog-ng OSE commits every log message to the target database individually. When the *explicit-commits* option is enabled, messages are committed in batches. This improves the performance, but results in some latency, as the messages are not immediately sent to the database. The size and frequency of batched commits can be set using the *flush_lines* and *flush_timeout* parameters. The *explicit-commits* option is available in syslog-ng OSE version 3.2 and later.

flush_lines()

Type: number

Default: Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type: time in milliseconds

Default: Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush_lines* option.

frac_digits()

Type: number

Default: 0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

host()

Type: hostname or IP address

Default: n/a

Description: Hostname of the database server. Note that Oracle destinations do not use this parameter, but retrieve the hostname from the /etc/tnsnames.ora file.

**Note**

If you specify `host="localhost"`, syslog-ng will use a socket to connect to the local database server. Use `host="127.0.0.1"` to force TCP communication between syslog-ng and the local database server.

To specify the socket to use, set and export the `MYSQL_UNIX_PORT` environment variable, for example `MYSQL_UNIX_PORT=/var/lib/mysql/mysql.sock; export MYSQL_UNIX_PORT`.

indexes()

Type: string list

Default: "date", "facility", "host", "program"

Description: The list of columns that are indexed by the database to speed up searching. To disable indexing for the destination, include the empty `indexes()` parameter in the destination, simply omitting the `indexes` parameter will cause syslog-ng to request indexing on the default columns.

The syslog-ng OSE application will create the name of indexes automatically with the following method:

- In case of MsSQL, PostgreSQL, MySQL or SQLite or (Oracle but tablename < 30 characters): `{table}_{column}_idx`.
- In case of Oracle and tablename > 30 characters: `md5sum of {table}_{column}-1` and the first character will be replaced by "i" character and the md5sum will be truncated to 30 characters.

local_time_zone()

Type: name of the timezone or the timezone offset

Default: The local timezone.

Description: Sets the timezone used when expanding filename and tablename templates. The timezone can be specified as using the name of the (for example `time_zone("Europe/Budapest")`), or as the timezone offset (for example `+01:00`). The valid timezone names are listed under the `/usr/share/zoneinfo` directory.

log_fifo_size()

Type: number

Default: Use global setting.

Description: The number of entries in the output buffer (output fifo).

null()

Type: string

Default:

Description: If the content of a column matches the string specified in the `null()` parameter, the contents of the column will be replaced with an SQL NULL value. If unset (by default), the option does not match on any string. For details, see the *Example 7.9, Using SQL NULL values (p. 105)*.

**Example 7.9. 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.

table()

Type: string
Default: messages

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

time_zone()

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



Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

type()

Type:	mssql, mysql, oracle, pgsql, or sqlite3
Default:	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 7.10. Value: default

```
columns("date datetime", "host varchar(32)", "row_id serial")
values("$R_DATE", "$HOST", default)
```

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

The *syslog()* driver sends messages to a remote host (for example a syslog-*ng* server or relay) on the local intranet or internet using the new standard syslog protocol developed by IETF (for details about the new protocol, see *Section 2.9.2, IETF-syslog messages (p. 12)*). The protocol supports sending messages using the UDP, TCP, or the encrypted TLS networking protocols.

The required arguments of the driver are the address of the destination host (where messages should be sent). The transport method (networking protocol) is optional, syslog-*ng* uses the TCP protocol by default. For the list of available optional parameters, see *Section 7.5.1, syslog() destination options (p. 107)*.

Declaration:

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

**Note**

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

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

**Note**

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

**Example 7.11. Using the syslog() driver**

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

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

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

Send the log messages using TLS encryption and use mutual authentication. For details on the encryption and authentication options, see *Section 10.4, TLS options (p. 151)*.

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

7.5.1. syslog() destination options

The *syslog()* driver sends messages to a remote host (for example a syslog-*ng* server or relay) on the local intranet or internet using the RFC5424 syslog protocol developed by IETF (for details about the protocol, see *Section 2.9.2, IETF-syslog messages (p. 12)*). The protocol supports sending messages using the UDP, TCP, or the encrypted TLS networking protocols.

These destinations have the following options:

flags()

Type:	no_multi_line, syslog-protocol
Default:	empty set

Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.



- *syslog-protocol*: The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type: number

Default: Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type: time in milliseconds

Default: Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush_lines* option.

frac_digits()

Type: number

Default: 0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

ip_tos()

Type: number

Default: 0

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

ip_ttl()

Type: number

Default: 0

Description: Specifies the Time-To-Live value of outgoing packets.

**keep-alive()**

Type: yes or no

Default: yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the *keep-alive* option is enabled for the source. When the *keep-alive* option is enabled, syslog-ng saves the contents of the output queue of the destination when receiving a HUP signal, reducing the risk of losing messages

localip()

Type: string

Default: 0.0.0.0

Description: The IP address to bind to before connecting to target.

localport()

Type: number

Default: 0

Description: The port number to bind to. Messages are sent from this port.

log_fifo_size()

Type: number

Default: Use global setting.

Description: The number of entries in the output buffer (output fifo).

port() or destport()

Type: number

Default: 601

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.

so_broadcast()

Type: yes or no

Default: no

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

**so_keepalive()**

Type: yes or no

Default: no

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

so_rcvbuf()

Type: number

Default: 0

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

so_sndbuf()

Type: number

Default: 0

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

spoof_source()

Type: yes or no

Default: no

Description: Enables source address spoofing. This means that the host running syslog-ng generates UDP packets with the source IP address matching the original sender of the message. It is useful when you want to perform some kind of preprocessing via syslog-ng then forward messages to your central log management solution with the source address of the original sender. This option only works for UDP destinations though the original message can be received by TCP as well. This option is only available if syslog-ng was compiled using the `--enable-spoof-source` configuration option.

suppress()

Type: seconds

Default: 0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type: string

Default: A format conforming to the default logfile format.



Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *syslogd* or *syslog-ng* itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type: yes or no

Default: no

Description: Turns on escaping for the ', ", and backspace characters in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type: number

Default: 0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying 0 or a lower value sets the output limit to unlimited.

time_zone()

Type: timezone in +/HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

tls()

Type: tls options

Default: n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations. TLS can be used only with the *tcp* transport protocols. For details, see *Section 10.4, TLS options (p. 151)*.

transport

Type: udp, tcp, or tls

Default: tcp

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



ts_format()

Type: rfc3164, bsd, rfc3339, iso

Default: rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. For details, see *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

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

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

All four drivers have a single required parameter specifying the destination host address, where messages should be sent. For the list of available optional parameters, see *Section 7.6.1, tcp(), tcp6(), udp(), and udp6() destination options (p. 112)*.

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

Declaration:

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



Example 7.12. Using the `tcp()` driver

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

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

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

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

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

(For details on how to use the IETF-syslog protocol, see *Section 7.5.1, syslog() destination options (p. 107)*.)

7.6.1. `tcp()`, `tcp6()`, `udp()`, and `udp6()` destination options

This driver sends messages to another host on the local intranet or internet according to RFC3164 using the UDP or TCP protocol. The `tcp6()` and `udp6()` drivers use the IPv6 network protocol.

These destinations have the following options:

flags()

Type: no_multi_line, syslog-protocol

Default: empty set



Description: Flags influence the behavior of the destination driver.

- *no-multi-line*: The *no-multi-line* flag disables line-breaking in the messages; the entire message is converted to a single line.
- *syslog-protocol*: The *syslog-protocol* flag instructs the driver to format the messages according to the new IETF syslog protocol standard. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type: number

Default: Use global setting

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

flush_timeout()

Type: time in milliseconds

Default: Use global setting

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush_lines* option.

frac_digits()

Type: number

Default: 0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

ip_tos()

Type: number

Default: 0

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

ip_ttl()

Type: number

Default: 0



Description: Specifies the Time-To-Live value of outgoing packets.

keep-alive()

Type: yes or no

Default: yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the `keep-alive` option is enabled for the source. When the `keep-alive` option is enabled, syslog-ng saves the contents of the output queue of the destination when receiving a HUP signal, reducing the risk of losing messages

localip()

Type: string

Default: 0.0.0.0

Description: The IP address to bind to before connecting to target.

localport()

Type: number

Default: 0

Description: The port number to bind to. Messages are sent from this port.

port() or destport()

Type: number

Default: 514

Description: The port number to connect to. Note that the default port numbers used by syslog-ng do not comply with the latest RFC which was published after the release of syslog-ng 3.0.2, therefore the default port numbers will change in the future releases.



Note

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

so_broadcast()

Type: yes or no

Default: no

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



so_keepalive()

Type: yes or no

Default: no

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

so_rcvbuf()

Type: number

Default: 0

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

so_sndbuf()

Type: number

Default: 0

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

spoof_source()

Type: yes or no

Default: no

Description: Enables source address spoofing. This means that the host running syslog-ng generates UDP packets with the source IP address matching the original sender of the message. It is useful when you want to perform some kind of preprocessing via syslog-ng then forward messages to your central log management solution with the source address of the original sender. This option only works for UDP destinations though the original message can be received by TCP as well. This option is only available if syslog-ng was compiled using the `--enable-spoof-source` configuration option.

suppress()

Type: seconds

Default: 0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type: string

Default: A format conforming to the default logfile format.



Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like *syslogd* or *syslog-ng* itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type: yes or no

Default: no

Description: Turns on escaping for the ', ", and backspace characters in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.

throttle()

Type: number

Default: 0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying 0 or a lower value sets the output limit to unlimited.

time_zone()

Type: timezone in +/HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

tls()

Type: tls options

Default: n/a

Description: This option sets various TLS specific options like key/certificate files and trusted CA locations. TLS can be used only with the *tcp* transport protocols. For details, see *Section 10.4, TLS options (p. 151)*.

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 *Section 2.5.1, A note on timezones and timestamps (p. 9)*.



7.7. Sending messages to UNIX domain sockets

The `unix-stream()` and `unix-dgram()` drivers send messages to a UNIX domain socket in either `SOCK_STREAM` or `SOCK_DGRAM` mode.

Both drivers have a single required argument specifying the name of the socket to connect to. For the list of available optional parameters, see *Section 7.7.1, unix-stream() and unix-dgram() destination options (p. 117)*.

Declaration:

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



Example 7.13. Using the unix-stream() driver

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

7.7.1. 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:

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. If this flag is enabled, macros used for the message have effect only for the text of the message, the message header is formatted to the new standard. Note that this flag is not needed for the *syslog* driver.

flush_lines()

Type:	number
Default:	Use global setting.

Description: Specifies how many lines are flushed to a destination at a time. Syslog-*ng* waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the destination, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.



flush_timeout()

Type: time in milliseconds

Default: Use global setting.

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For details, see the *flush_lines* option.

frac_digits()

Type: number

Default: 0

Description: The syslog-ng application can store fractions of a second in the timestamps according to the ISO8601 format.. The *frac_digits()* parameter specifies the number of digits stored. The digits storing the fractions are padded by zeros if the original timestamp of the message specifies only seconds. Fractions can always be stored for the time the message was received. Note that syslog-ng can add the fractions to non-ISO8601 timestamps as well.

log_fifo_size()

Type: number

Default: Use global setting.

Description: The number of entries in the output buffer (output fifo).

keep-alive()

Type: yes or no

Default: yes

Description: Specifies whether connections to destinations should be closed when syslog-ng is restarted (upon the receipt of a SIGHUP signal). Note that this applies to the client (destination) side of the syslog-ng connections, server-side (source) connections are always reopened after receiving a HUP signal unless the *keep-alive* option is enabled for the source. When the *keep-alive* option is enabled, syslog-ng saves the contents of the output queue of the destination when receiving a HUP signal, reducing the risk of losing messages

so_broadcast()

Type: yes or no

Default: no

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

so_keepalive()

Type: yes or no

Default: no



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

so_rcvbuf()

Type:	number
Default:	0

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

so_sndbuf()

Type:	number
Default:	0

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

suppress()

Type:	seconds
Default:	0 (disabled)

Description: If several identical log messages would be sent to the destination without any other messages between the identical messages (for example, an application repeated an error message ten times), syslog-ng can suppress the repeated messages and send the message only once, followed by the *Last message repeated n times.* message. The parameter of this option specifies the number of seconds syslog-ng waits for identical messages.

template()

Type:	string
Default:	A format conforming to the default logfile format.

Description: Specifies a template defining the logformat to be used in the destination. Macros are described in *Section 11.1.3, Macros of syslog-ng OSE (p. 155)*. Please note that for network destinations it might not be appropriate to change the template as it changes the on-wire format of the syslog protocol which might not be tolerated by stock syslog receivers (like `syslogd` or `syslog-ng` itself). For network destinations make sure the receiver can cope with the custom format defined.

template_escape()

Type:	yes or no
Default:	no

Description: Turns on escaping for the ', ", and backspace characters in templated output files. This is useful for generating SQL statements and quoting string contents so that parts of the log message are not interpreted as commands to the SQL server.



throttle()

Type: number

Default: 0

Description: Sets the maximum number of messages sent to the destination per second. Use this output-rate-limiting functionality only when using disk-buffer as well to avoid the risk of losing messages. Specifying 0 or a lower value sets the output limit to unlimited.

time_zone()

Type: timezone in +/HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

ts_format()

Type: rfc3164, bsd, rfc3339, iso

Default: rfc3164

Description: Override the global timestamp format (set in the global `ts_format()` parameter) for the specific destination. For details, see *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

7.8. Sending messages to a user terminal — usertty() destination

This driver writes messages to the terminal of a logged-in user.

The `usertty()` driver has a single required argument, specifying a username who should receive a copy of matching messages. Use the asterisk * to specify every user currently logged in to the system.

Declaration:

```
usertty(username);
```

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



Example 7.14. Using the usertty() driver

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



Chapter 8. Routing messages: log paths and filters

8.1. 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:

```
log {
    source(s1); source(s2); ...
    optional_element(filter1|parser1|rewrite1);
optional_element(filter2|parser2|rewrite2);
    destination(d1); destination(d2); ...
    flags(flag1[, flag2...]);
};
```



Warning

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



Example 8.1. A simple log statement

The following log statement sends all messages arriving to the localhost to a remote server.

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

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

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

- *final*: Do not send the messages processed by this log path to any further destination.
- *fallback*: Process messages that were not processed by other log paths.
- *catchall*: Process every message, regardless of its source or if it was already processed by other log paths.
- *flow-control*: Stop reading messages from the source if the destination cannot accept them. For details, see *Section 8.2, Managing incoming and outgoing messages with flow-control (p. 125)*.

For details on the individual flags, see *Section 8.1.2, Log path flags (p. 124)*. The effect and use of the *flow-control* flag is detailed in *Section 8.2, Managing incoming and outgoing messages with flow-control (p. 125)*.



8.1.1. Embedded log statements

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

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

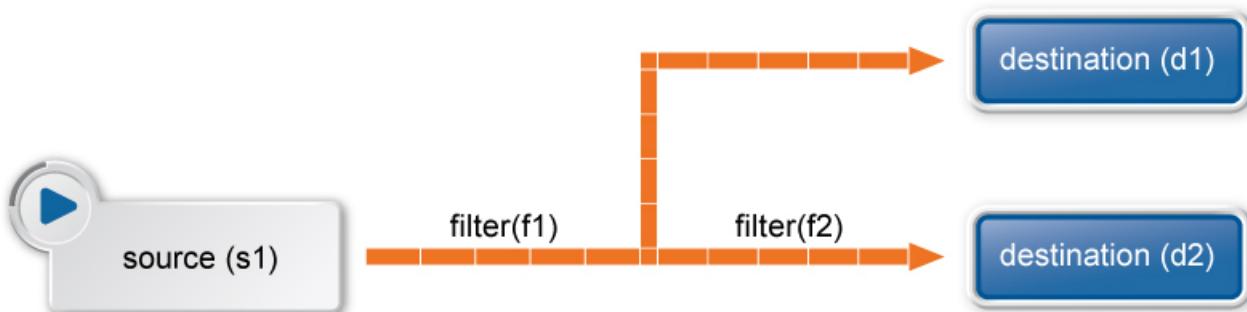


Figure 8.1. Embedded log statement

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

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

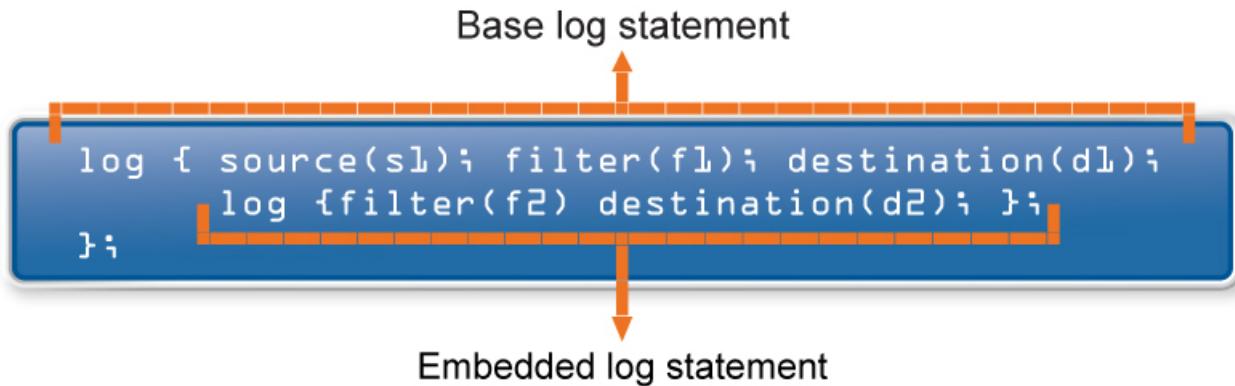


Figure 8.2. Embedded log statements

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

8.1.1.1. Using embedded log statements

Embedded log statements (for details, see *Section 8.1.1, Embedded log statements (p. 122)*) re-use the results of processing messages (for example the results of filtering or rewriting) to create complex log paths. Embedded log statements use the same syntax as regular log statements, but they cannot contain additional sources. To define embedded log statements, use the following syntax:

```

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

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

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

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

**Example 8.2. Using embedded log paths**

The following log path sends every message to the `d_file1` and the `d_file2` destinations.

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

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

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

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

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

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

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

8.1.2. Log path flags

Flags influence the behavior of syslog-ng, and the way it processes messages. The following flags may be used in the log paths, as described in *Section 8.1, Log paths (p. 121)*.

Flag	Description
<code>catchall</code>	This flag means that the source of the message is ignored, only the filters are taken into account when matching messages. A log statement using the <code>catchall</code> flag processes every message that arrives to any of the defined sources.
<code>fallback</code>	This flag makes a log statement 'fallback'. Fallback log statements process messages that were not processed by other, 'non-fallback' log statements.
<code>final</code>	This flag means that the processing of log messages processed by the log statement ends here, other log statements appearing later in the configuration file will not process the messages processed by the log statement labeled as 'final'. Note that this does not necessarily mean that matching messages will be stored only once, as there can be matching log statements processed prior the current one.
<code>flow-control</code>	Enables flow-control to the log path, meaning that syslog-ng will stop reading messages from the sources of this log statement if the destinations are not able to process the messages at the required speed. If disabled, syslog-ng will drop messages if the destination queues are full. If enabled, syslog-ng will only drop messages if the destination queues/window sizes are improperly sized. For details, see <i>Section 8.2, Managing incoming and outgoing messages with flow-control (p. 125)</i> .

Table 8.1. Log statement flags

**Warning**

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

**Example 8.3. Using log path flags**

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

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

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

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

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

8.2. 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. To use flow-control, the *flow-control* flag must be enabled for the particular log path.

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

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

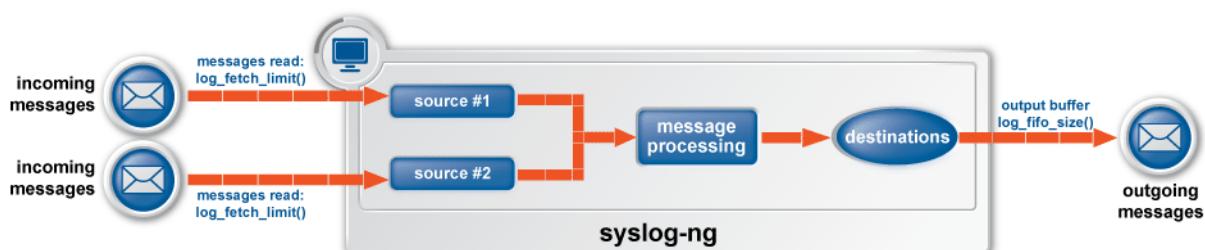


Figure 8.3. Managing log messages in syslog-*ng*

**Note**

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

Every destination has its own output buffer. The output buffer is needed because the destination might not be able to accept all messages immediately. The *log_fifo_size()* parameter sets the size of the output buffer. The



output buffer must be larger than the `log_fetch_limit()` of the sources, to ensure that every message read during the poll loop fits into the output buffer. If the log path sends messages to a destination from multiple sources, the output buffer must be large enough to store the incoming messages of every source.

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

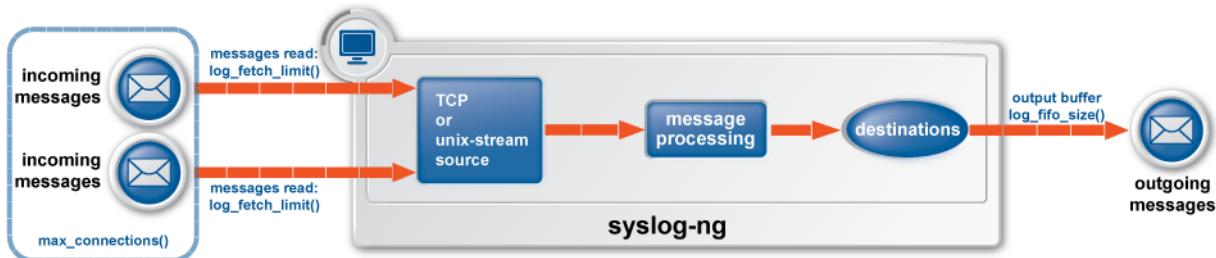


Figure 8.4. Managing log messages of TCP sources in syslog-ng

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

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

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

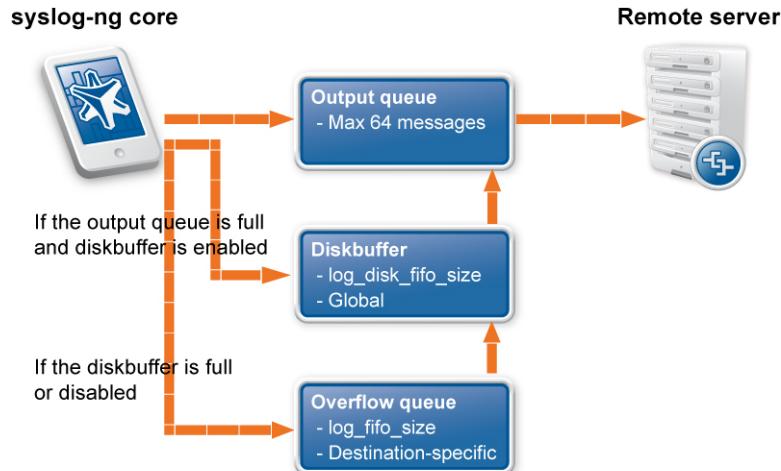


Figure 8.5. Handling outgoing messages in syslog-ng PE

- **Output queue:** Messages from the output queue are sent to the target syslog-ng server. The syslog-ng application puts the outgoing messages directly into the output queue, unless the output queue is full. The output queue can hold 64 messages, this is a fixed value and cannot be modified.
- **Disk buffer:** If the output queue is full and disk-buffering is enabled, syslog-ng Premium Edition puts the outgoing messages into the disk buffer of the destination.
- **Overflow queue:** If the output queue is full and the disk buffer is disabled or full, syslog-ng puts the outgoing messages into the overflow queue of the destination. (The overflow queue is identical to the output buffer used by other destinations.) The `log_fifo_size()` parameter specifies the number of messages stored in the overflow queue. For details on sizing the `log_fifo_size()` parameter, see *Section 8.2, Managing incoming and outgoing messages with flow-control (p. 125)*.

8.2.1. 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 syslog-ng stops reading the source. For example, if messages from a source are sent to a remote server and also stored locally in a file, and the network connection to the server becomes unavailable, neither the remote server nor the local file will receive any messages.



Note

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

8.2.2. Configuring flow-control

For details on how flow-control works, see *Section 8.2, Managing incoming and outgoing messages with flow-control (p. 125)*. The summary of the main points is as follows:



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

**Note**

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

**Example 8.4. Sizing parameters for flow-control**

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

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

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

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

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



For details, see also *Section 16.2, Handling lots of parallel connections (p. 196)*.

8.3. Filters

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

- *Section 8.3.1, Using filters (p. 129)* describes how to configure and use filters.
- *Section 8.3.2, Combining filters with boolean operators (p. 129)* shows how to create complex filters using boolean operators.
- *Section 8.3.3, Comparing macro values in filters (p. 130)* explains how to evaluate macros in filters.
- *Section 8.3.4, Using wildcards, special characters, and regular expressions in filters (p. 131)* provides tips on using regular expressions.
- *Section 8.3.5, Tagging messages (p. 132)* explains how to tag messages and how to filter on the tags.
- *Section 8.3.6, Filter functions (p. 132)* is a detailed description of the filter functions available in syslog-ng OSE.

8.3.1. Using filters

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

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

```
filter <identifier> { <filter_type>("<filter_expression>"); };
```



Example 8.5. 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"); };
```

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

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

8.3.2. Combining filters with boolean operators

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

```
filter demo_filter1 { host("example1"); };
filter demo_filter2 { host("example2"); };
log demo_filteredlog {
    source(s1); source(s2);
```



```
filter(demo_filter1); filter(demo_filter2);
destination(d1); destination(d2); };
```

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

```
filter demo_filter { host("example1") or host("example2"); };
log demo_filteredlog {
    source(s1); source(s2);
    filter(demo_filter);
    destination(d1); destination(d2); };
```

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

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

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

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

Alternatively, you can use parentheses to avoid this confusion:

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

For a complete description on filter functions, see *Section 8.3.6, Filter functions (p. 132)*.

The following filter statement selects the messages that contain the word `deny` and come from the host `example`.

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

The `value()` parameter of the `match` function limits the scope of the function to the text part of the message (that is, the part returned by the `$MESSAGE` macro). For details on using the `match()` filter function, see *Section match() (p. 134)*.



Tip

Filters are often used together with log path flags. For details, see *Section 8.1.2, Log path flags (p. 124)*.

8.3.3. Comparing macro values in filters

Starting with syslog-ng OSE version 3.2, it is also possible to compare macro values and templates as numerical and string values. String comparison uses alphabetical: it determines if a string is alphabetically greater or equal to another string. Use the following syntax to compare macro values or templates. For details on macros and templates, see *Section 11.1, Customizing message format (p. 153)*.

```
filter <filter-id>
    {"<macro-or-template>" operator "<value-or-macro-or-template>"};
```

**Example 8.6. 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 accomplishes the same, but uses a template as the left argument of the operator and compares the values as strings:

```
filter f_pid {"$HOST$PID" eq "$HOST"};
```

The following example selects messages with priority level 4 or higher.

```
filter f_level {"$LEVEL_NUM" > "5"};
```

Note that:

- The macro or template must be enclosed in double-quotes.
- The \$ character must be used before macros.
- Using comparator operators can be equivalent to using filter functions, but is somewhat slower. For example, using "\$HOST" eq "myhost" is equivalent to using host ("myhost" type(string)).
- You can use any macro in the expression, including user-defined macros from parsers and results of pattern database classifications.
- The results of filter functions are boolean values, so they cannot be compared to other values.
- You can use boolean operators to combine comparison expressions.

The following operators are available:

Numerical operator	String operator	Meaning
==	eq	Equals
!=	ne	Not equal to
>	gt	Greater than
<	lt	Less than
>=	ge	Greater than or equal
=<	le	Less than or equal

Table 8.2. Numerical and string comparison operators

8.3.4. 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, syslog-ng OSE uses POSIX 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 8.7. 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));};
```

For details on using regular expressions in syslog-ng OSE, see *Section 11.3, Regular expressions (p. 164)*.

To filter for special control characters like the carriage return (CR), use the `\r` escape prefix in syslog-ng OSE version 3.0 and 3.1. In syslog-ng OSE 3.2 and later, you can also use the `\x` escape prefix and the ASCII code of the character. For example, to filter on carriage returns, use the following filter:

```
filter f_carriage_return {match("\x0d" value ("MESSAGE"));};
```

8.3.5. Tagging messages

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

- at the source when the message is received; and
- when the message matches a pattern in the pattern database. For details on using the pattern database, see *Section 13.2, Using pattern databases (p. 176)*, for details on creating tags in the pattern database, see *Section 13.5.3, The syslog-ng pattern database format (p. 184)*.

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

**Note**

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

For an example on tagging, see *Example 8.8, Adding tags and filtering messages with tags (p. 136)*.

8.3.6. Filter functions

The following functions may be used in the filter statement, as described in *Section 8.3, Filters (p. 129)*.

Name	Description
<code>facility()</code>	Filter messages based on the sending facility.
<code>filter()</code>	Call another filter function.
<code>host()</code>	Filter messages based on the sending host.
<code>level() or priority()</code>	Filter messages based on their priority.



Name	Description
<code>match()</code>	Use a regular expression to filter messages based on a specified header or content field.
<code>message()</code>	Use a regular expression to filter messages based their content.
<code>netmask()</code>	Filter messages based on the IP address of the sending host.
<code>program()</code>	Filter messages based on the sending application.
<code>source()</code>	Select messages of the specified syslog-ng OSE source statement.
<code>tags()</code>	Select messages having the specified tag.

Table 8.3. Filter functions available in syslog-ng OSE

facility()Synopsis: `facility(facility[,facility])`

Description: Match messages having one of the listed facility code. An alternate syntax permits the use an arbitrary facility codes.

The `facility()` filter accepts both the name and the numerical code of the facility or the importance level.

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

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

The syslog-ng application recognizes the following facilities: (Note that some of these facilities are available only on specific platforms.)

Numerical Code	Facility name	Facility
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news subsystem
8	uucp	UUCP subsystem
9	cron	clock daemon
10	auth	security/authorization messages
11	ftp	FTP daemon
12		NTP subsystem
13		log audit



Numerical Code	Facility name	Facility
14		log alert
15	cron	clock daemon
16-23	local0..local7	locally used facilities (local0-local7)

Table 8.4. syslog Message Facilities recognized by the facility() filter

facility()

Synopsis: facility(<numeric facility code>)

Description: An alternate syntax for *facility* permitting the use of an arbitrary facility code. Facility codes 0-23 are predefined and can be referenced by their usual name. Facility codes above 24 are not defined but can be used by this alternate syntax.

filter()

Synopsis: filter(filtername)

Description: Call another filter rule and evaluate its value.

host()

Synopsis: host(regexp)

Description: Match messages by using a regular expression against the hostname field of log messages.

level() or priority()

Synopsis: level(pri[,pri1..pri2[,pri3]])

Description: Match messages based on priority.

The *level()* filter accepts the following levels: *emerg*, *alert*, *crit*, *err*, *warning*, *notice*, *info*, *debug*.

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

```
level(warning)
```

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

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

match()

Synopsis: match(regexp)



Description: Match a regular expression to the headers and the message itself (that is, the values returned by the *MSGHDR* and *MSG* macros). Note that in syslog-ng version 2.1 and earlier, the *match()* filter was applied only to the text of the message, excluding the headers. This functionality has been moved to the *message()* filter.

To limit the scope of the match to a specific part of the message (identified with a macro), use the *match(regexp value("MACRO"))* syntax. Do not include the \$ sign in the parameter of the *value()* option.

The *value()* parameter accepts both built-in macros and user-defined ones created with a parser or using a pattern database. For details on macros and parsers, see *Section 11.1.2, Templates and macros (p. 154)*, *Section 12.1, Parsing messages (p. 168)*, and *Section 13.2.1, Using parser results in filters and templates (p. 177)*.

message()

Synopsis: `message(regexp)`

Description: Match a regular expression to the text of the log message, excluding the headers (that is, the value returned by the *MSG* macros). Note that in syslog-ng version 2.1 and earlier, this functionality was performed by the *match()* filter.

netmask()

Synopsis: `netmask(ip/mask)`

Description: Select only messages sent by a host whose IP address belongs to the specified IP subnet. Note that this filter checks the IP address of the last-hop relay (the host that actually sent the message to syslog-ng), not the contents of the *HOST* field of the message.

program()

Synopsis: `program(regexp)`

Description: Match messages by using a regular expression against the program name field of log messages.

source()

Synopsis: `source id`

Description: Select messages of a source statement. This filter can be used in embedded log statements if the parent statement contains multiple source groups — only messages originating from the selected source group are sent to the destination of the embedded log statement.

tags()

Synopsis: `tag`

Description: Select messages labeled with the specified tag. Every message automatically has the tag of its source in *.source.<id_of_the_source_statement>* format. This option is available only in syslog-ng 3.1 and later.

**Example 8.8. Adding tags and filtering messages with tags**

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

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

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

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

Starting with version 3.2, syslog-ng OSE 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");};
```

8.4. Dropping messages

To skip the processing of a message without sending it to a destination, create a log statement with the appropriate filters, but do not include any destination in the statement, and use the `final` flag.

**Example 8.9. Skipping messages**

The following log statement drops all `debug` level messages without any further processing.

```
filter demo_debugfilter { level(debug); };
log { source(s_all); filter(demo_debugfilter); flags(final); };
```



Chapter 9. Global options of syslog-*ng* OSE

9.1. Configuring global syslog-*ng* options

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

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



Example 9.1. Using global options

To disable domain name resolving, add the following line to the syslog-*ng* configuration file:

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

For a detailed list of the available options, see *Section 9.2, “Global options (p. 137)*. For important global options and recommendations on their use, see *Chapter 16, “Best practices and examples (p. 196)*.

9.2. Global options

The following options can be specified in the options statement, as described in *Section 9.1, “Configuring global syslog-*ng* options (p. 137)*.

bad_hostname()

Accepted values:	regular expression
Default:	no

Description: A regexp containing hostnames which should not be handled as hostnames.

chain_hostnames()

Accepted values:	yes no
Default:	no

Description: Enable or disable the chained hostname format.

check_hostname()

Accepted values:	yes no
Default:	no

Description: Enable or disable checking whether the hostname contains valid characters.

**create_dirs()**

Accepted values: yes | no

Default: no

Description: Enable or disable directory creation for destination files.**dir_group()**

Accepted values: groupid

Default: root

Description: The default group for newly created directories.**dir_owner()**

Accepted values: userid

Default: root

Description: The default owner of newly created directories.**dir_perm()**

Accepted values: permission value

Default: 0700

Description: The permission mask of directories created by syslog-ng. Log directories are only created if a file after macro expansion refers to a non-existing directory, and directory creation is enabled (see also the `create_dirs()` option). For octal numbers prefix the number with `0`, for example use `0755` for `rwxr-xr-x`.To preserve the original properties of an existing directory, use the option without specifying an attribute: `dir_perm()`. Note that when creating a new directory without specifying attributes for `dir_perm()`, the default permission of the directories is masked with the umask of the parent process (typically `0022`).**dns_cache()**

Accepted values: yes | no

Default: yes

Description: Enable or disable DNS cache usage.**dns_cache_expire()**

Accepted values: number

Default: 3600

Description: Number of seconds while a successful lookup is cached.



[dns_cache_expire_failed\(\)](#)

Accepted values: number

Default: 60

Description: Number of seconds while a failed lookup is cached.

[dns_cache_hosts\(\)](#)

Accepted values: filename

Default: unset

Description: Name of a file in /etc/hosts format that contains static IP->hostname mappings. Use this option to resolve hostnames locally without using a DNS. Note that any change to this file triggers a reload in syslog-ng and is instantaneous.

[dns_cache_size\(\)](#)

Accepted values: number

Default: 1007

Description: Number of hostnames in the DNS cache.

[time_zone\(\)](#)

Type: timezone in +/HH:MM format

Default: unspecified

Description: Convert timestamps to the timezone specified by this option. If this option is not set then the original timezone information in the message is used.

[flush_lines\(\)](#)

Accepted values: number

Default: 0

Description: Specifies how many lines are flushed to a destination at a time. Syslog-ng waits for this number of lines to accumulate and sends them off in a single batch. Setting this number high increases throughput as fully filled frames are sent to the network, but also increases message latency. The latency can be limited by the use of the *flush_timeout* option.

[flush_timeout\(\)](#)

Accepted values: time in milliseconds

Default: 10000

Description: Specifies the time syslog-ng waits for lines to accumulate in its output buffer. For more information, see the *flush_lines()* option.



group()

Accepted values: groupid

Default: root

Description: The default group of output files. By default, syslog-ng changes the privileges of accessed files (for example `/dev/null`) to `root.root 0600`. To disable modifying privileges, use this option with the `-1` value.

keep_hostname()

Type: yes or no

Default: no

Description: Enable or disable hostname rewriting.

- If enabled (`keep_hostname (yes)`), syslog-ng OSE assumes that the incoming log message was sent by the host specified in the `HOST` field of the message.
- If disabled (`keep_hostname (no)`), syslog-ng OSE rewrites the `HOST` field of the message, either to the IP address (if the `use_dns ()` parameter is set to `no`), or to the hostname (if the `use_dns ()` parameter is set to `yes` and the IP address can be resolved to a hostname) of the host sending the message to syslog-ng OSE. For details on using name resolution in syslog-ng OSE, see *Section 16.4, Using name resolution in syslog-ng* (p. 197).



Note

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

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

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



Note

When relaying messages, enable this option on the syslog-ng OSE server and also on every relay, otherwise syslog-ng OSE will treat incoming messages as if they were sent by the last relay.

keep_timestamp()

Type: yes or no

Default: yes

Description: Specifies whether syslog-ng should accept the timestamp received from the sending application or client. If disabled, the time of reception will be used instead. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.

**log_fifo_size()**

Accepted values:	number
Default:	1000

Description: The number of lines fitting to the output queue. Note that it is not possible to set this option lower than 1000.

log_msg_size()

Accepted values:	number
Default:	8192

Description: Maximum length of a message in bytes.

normalize_hostnames()

Accepted values:	yes no
Default:	no

Description: Normalize hostnames, which currently translates to converting them to lower case. (requires 1.9.9)

owner()

Accepted values:	userid
Default:	root

Description: The default owner of output files. By default, syslog-ng changes the privileges of accessed files (for example `/dev/null`) to `root.root 0600`. To disable modifying privileges, use this option with the `-1` value.

mark()

Accepted values:	number
Default:	1200

Description: An alias for the obsolete `mark_freq()` option, retained for compatibility with syslog-ng version 1.6.x.

mark_freq()

Accepted values:	number
Default:	1200

Description: The number of seconds between two `MARK` messages. `MARK` messages are generated when there was no message traffic to inform the receiver that the connection is still alive. Note that only local messages postpone the sending of the `MARK` message, relayed messages do not. If set to zero (0), no `MARK` messages are sent.



perm()

Accepted values: permission value

Default: 0600

Description: The default permission for output files. By default, syslog-ng changes the privileges of accessed files (for example `/dev/null`) to `root.root 0600`. To disable modifying privileges, use this option with the `-1` value.

recv_time_zone()

Accepted values: time offset (for example: `+03:00`)

Default: local timezone

Description: Specifies the time zone associated with the incoming messages, if not specified otherwise in the message or in the source driver. For details, see also *Section 2.5, Timezones and daylight saving (p. 8)* and *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

send_time_zone()

Accepted values: time offset (for example: `+03:00`)

Default: local timezone

Description: Specifies the time zone associated with the messages sent by syslog-ng, if not specified otherwise in the message or in the destination driver. For details, see *Section 2.5, Timezones and daylight saving (p. 8)*.

stats_freq()

Accepted values: number

Default: 600

Description: The period between two STATS messages in seconds. STATS are log messages sent by syslog-ng, containing statistics about dropped log messages. Set to `0` to disable the STATS messages.

stats_level()

Accepted values: 0 | 1 | 2 | 3

Default: 0

Description: Specifies the detail of statistics syslog-ng collects about the processed messages.

- Level 0 collects only statistics about the sources and destinations
- Level 1 contains details about the different connections and log files, but has a slight memory overhead
- Level 2 contains detailed statistics based on the hostname.
- Level 3 contains detailed statistics based on various message parameters like facility, severity, or tags.

Note that level 2 and 3 increase the memory requirements and CPU load. For details on message statistics, see *Chapter 14, Statistics of syslog-ng (p. 191)*.



sync() or sync_freq() (DEPRECATED)

Accepted values: number

Default: 0

Description: Obsolete aliases for *flush_lines()*

time_reap()

Accepted values: number

Default: 60

Description: The time to wait in seconds before an idle destination file is closed. Note that only destination files having macros in their filenames are closed automatically.

time_reopen()

Accepted values: number

Default: 60

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

time_sleep()

Accepted values: number

Default: 0

Description: The time to wait in milliseconds between each invocation of the *poll()* iteration.

ts_format()

Accepted values: rfc3164 | bsd | rfc3339 | iso

Default: rfc3164

Description: Specifies the timestamp format used when syslog-ng itself formats a timestamp and nothing else specifies a format (for example: *STAMP* macros, internal messages, messages without original timestamps). For details, see also *Section 2.5.1, A note on timezones and timestamps (p. 9)*.

use_dns()

Type: yes, no, persist_only

Default: yes

Description: Enable or disable DNS usage. The *persist_only* option attempts to resolve hostnames locally from file (for example from */etc/hosts*). syslog-ng blocks on DNS queries, so enabling DNS may lead to a Denial of Service attack. To prevent DoS, protect your syslog-ng network endpoint with firewall rules, and make sure that all hosts which may get to syslog-ng are resolvable. This option can be specified globally, and per-source as well. The local setting of the source overrides the global option if available.



use_fqdn()

Type: yes or no

Default: no

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

use_time_recv() (DEPRECATED)

Accepted values: yes | no

Default: no



Warning

This option is not available in syslog-ng OSE version 3.2 and later. Use the *R_* prefixed version of the respective macro instead. Starting with syslog-ng OSE version 3.2, the *DATE* macro equals the *S_DATE* macro.

Description: This option controls how the time related macros are expanded in filename and content templates. If set to yes, then the non-prefixed versions of the time related macros (for example: *HOUR* instead of *R_HOUR* and *S_HOUR*) refer to the time when the message was received, otherwise it refers to the timestamp which is in the message.



Note

The timestamps in the messages are generated by the originating host and might not be accurate.

This option is deprecated as many users assumed that it controls the timestamp as it is written to logfiles/destinations, which is not the case. To change how messages are formatted, specify a content-template referring to the appropriate prefixed (*S_* or *R_*) time macro.



Chapter 10. TLS-encrypted message transfer

10.1. Secure logging using TLS

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

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



Figure 10.1. Certificate-based authentication

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

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

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

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

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

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

For details on configuring TLS communication in syslog-ng, see *Section 10.2, “Encrypting log messages with TLS (p. 146)*.



10.2. 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 *Section 10.1, Secure logging using TLS* (p. 145).

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



Note

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

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

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

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

Purpose:

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

Steps:

Step 1. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng server to the syslog-ng client hosts, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.

Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem`. The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

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

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

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



Example 10.1. A destination statement using TLS

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

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

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



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

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


Warning

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

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

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

Purpose:

Complete the following steps on the syslog-ng server:

Steps:

- Step 1. Copy the certificate (for example `syslog-ng.cert`) of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/cert.d` directory. The certificate must be a valid X.509 certificate in PEM format.
- Step 2. Copy the private key (for example `syslog-ng.key`) matching the certificate of the syslog-ng server to the syslog-ng server host, for example into the `/opt/syslog-ng/etc/syslog-ng/key.d` directory. The key must be in PEM format, and must not be password-protected.
- Step 3. Add a source statement to the syslog-ng configuration file that uses the `tls(key_file(key_file_fullpathname) cert_file(cert_file_fullpathname))` option and specify the key and certificate files. The source must use the source driver (`tcp()` or `tcpv6()`) matching the destination driver used by the syslog-ng client.


Example 10.2. A source statement using TLS

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

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

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

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



Step 4. Disable mutual authentication for the source by setting the following TLS option in the source statement:

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

For details on how to configure mutual authentication, see *Section 10.3, Mutual authentication using TLS (p. 148)*.



Example 10.3. Disabling mutual authentication

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

```
source demo_tls_source {
    tcp(ip(0.0.0.0) port(1999)
        tls( key_file("/opt/syslog-ng/etc/syslog-ng/key.d/syslog-ng.key")
            cert_file("/opt/syslog-ng/etc/syslog-ng/cert.d/syslog-ng.cert")
            peer_verify(optional-untrusted)) );};
```

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

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



Warning

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

For the details of the available `tls()` options, see *Section 10.4, TLS options (p. 151)*.

10.3. Mutual authentication using TLS

This section describes how to configure mutual authentication between the syslog-ng server and the client. Configuring mutual authentication is similar to configuring TLS (for details, see *Section 10.2, Encrypting log messages with TLS (p. 146)*), but the server verifies the identity of the client as well. Therefore, each client must have a certificate, and the server must have the certificate of the CA that issued the certificate of the clients. For the concepts of using TLS in syslog-ng, see *Section 10.1, Secure logging using TLS (p. 145)*.

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

Purpose:

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

Steps:

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



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

Step 3. Copy the CA certificate of the Certificate Authority (for example `cacert.pem`) that issued the certificate of the syslog-ng server to the syslog-ng client hosts, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.

Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem`. The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

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

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

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



Example 10.4. A destination statement using mutual authentication

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

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

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

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



Warning

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

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



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

Purpose:

Complete the following steps on the syslog-ng server:

Steps:

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

Step 2. Copy the CA certificate (for example `cacert.pem`) of the Certificate Authority that issued the certificate of the syslog-ng clients to the syslog-ng server, for example into the `/opt/syslog-ng/etc/syslog-ng/ca.d` directory.

Issue the following command on the certificate: `openssl x509 -noout -hash -in cacert.pem`
The result is a hash (for example `6d2962a8`), a series of alphanumeric characters based on the Distinguished Name of the certificate.

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

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

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

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



Example 10.5. A source statement using TLS

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

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

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

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

**Warning**

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

For the details of the available `tls()` options, see *Section 10.4, “TLS options” (p. 151)*.

10.4. TLS options

The syslog-ng application is able to encrypt incoming and outgoing syslog message flows using SSL/TLS, if the TCP transport protocol (the `tcp()` or `tcp6()` sources or destination) is used.

**Note**

The format of the TLS connections used by syslog-ng is similar to using syslog-ng and stunnel, but the source IP information is not lost.

To encrypt connections, use the `tls()` option in the source and destination statements.

The `tls()` option can include the following settings:

ca_dir()

Accepted values:	Directory name
Default:	none

Description: Name of a directory, that contains a set of trusted CA certificates in PEM format. The CA certificate files has to be named after the 32-bit hash of the subject's name. This naming can be created using the `c_rehash` utility in openssl.

cert_file()

Accepted values:	Filename
Default:	none

Description: Name of a file, that contains an X.509 certificate in PEM format, suitable as a TLS certificate, matching the private key.

crl_dir()

Accepted values:	Directory name
Default:	none

Description: Name of a directory that contains the Certificate Revocation Lists for trusted CAs. Similarly to `ca_dir()` files, use the 32-bit hash of the name of the issuing CAs as filenames. The extension of the files must be `.r0`.



key_file()

Accepted values: Filename

Default: none

Description: Name of a file, that contains an unencrypted private key in PEM format, suitable as a TLS key.

peer_verify()

Accepted values: optional-trusted | optional-untrusted | required-trusted | required-untrusted

Default: required-trusted

Description: Verification method of the peer, the four possible values is a combination of two properties of validation: whether the peer is required to provide a certificate (required or optional prefix), and whether the certificate provided needs to be trusted or not. For untrusted certificates only the existence of the certificate is checked, but it does not have to be valid — syslog-ng accepts the certificate even if it is expired, signed by an unknown CA, or its CN and the name of the machine mismatch.

trusted_dn()

Accepted values: list of accepted distinguished names

Default: none

Description: To accept connections only from hosts using certain certificates signed by the trusted CAs, list the distinguished names of the accepted certificates in this parameter. For example using `trusted_dn("*, O=Example Inc, ST=Some-State, C=*)` will accept only certificates issued for the *Example Inc* organization in *Some-State* state.

trusted_keys()

Accepted values: list of accepted SHA-1 fingerprints

Default: none

Description: To accept connections only from hosts using certain certificates having specific SHA-1 fingerprints, list the fingerprints of the accepted certificates in this parameter. For example `trusted_keys("SHA1:00:EF:ED:A4:CE:00:D1:14:A4:AB:43:00:EF:00:91:85:FF:89:28:8F", "SHA1:0C:42:00:3E:B2:60:36:64:00:E2:83:F0:80:46:AD:00:A8:9D:00:15")`.



Note

When using the `trusted_keys()` and `trusted_dn()` parameters, note the following:

- First, the `trusted_keys()` parameter is checked. If the fingerprint of the peer is listed, the certificate validation is performed.
- If the fingerprint of the peer is not listed in the `trusted_keys()` parameter, the `trusted_dn()` parameter is checked. If the DN of the peer is not listed in the `trusted_dn()` parameter, the authentication of the peer fails and the connection is closed.



Chapter 11. Manipulating messages

This chapter explains the methods that you can use to customize, reformat, and modify log messages using syslog-*ng* Open Source Edition.

- *Section 11.1, Customizing message format (p. 153)* explains how to use templates and macros to change the format of log messages, or the names of logfiles and database tables.
- *Section 11.2, Modifying messages (p. 162)* describes how to use rewrite rules to search and replace certain parts of the message content.
- *Section 11.3, Regular expressions (p. 164)* lists the different types of regular expressions that can be used in various syslog-*ng* OSE objects like filters and rewrite rules.

11.1. Customizing message format

The following sections describe how to customize the names of logfiles, and also how to use templates, macros, and template functions.

- *Section 11.1.1, Formatting messages, filenames, directories, and tablenames (p. 153)* explains how macros work.
- *Section 11.2, Modifying messages (p. 162)* describes how to use macros and templates to format log messages or change the names of logfiles and database tables.
- *Section 11.1.3, Macros of syslog-*ng* OSE (p. 155)* lists the different types of macros available in syslog-*ng* OSE.
- *Section 11.1.4, Using template functions (p. 160)* explains what template functions are and how to use them.
- *Section 11.1.5, Template functions of syslog-*ng* OSE (p. 160)* lists the template functions available in syslog-*ng* OSE.

11.1.1. Formatting messages, filenames, directories, and tablenames

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

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

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

- For details on using templates and macros, see *Section 11.1.2, Templates and macros (p. 154)*.
- For a list and description of the macros available in syslog-*ng* OSE, see *Section 11.1.3, Macros of syslog-*ng* OSE (p. 155)*.



- For details on using custom macros created with CSV parsers and pattern databases, see *Chapter 12, Parsing and segmenting structured messages* (p. 168) and *Section 13.2.1, Using parser results in filters and templates* (p. 177), respectively.

11.1.2. Templates and macros

The syslog-ng OSE application allows you to define message templates, and reference them from every object that can use a template. Templates can be used for example to create standard message formats or filenames. Templates can reference one or more macros (for example date, the hostname, and so on). For a list of macros available in syslog-ng Open Source Edition, see *Section 11.1.3, Macros of syslog-ng OSE* (p. 155). Fields from the structured data (SD) part of messages using the new IETF-syslog standard can also be used as macros.

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

**Note**

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

Macros can be included by prefixing the macro name with a `$` sign, just like in Bourne compatible shells. Although using braces around macro names is not mandatory, and the `"$MSG"` and `"${MSG}"` formats are equivalent, using the `"${MSG}"` format is recommended for clarity.

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}
```

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

Starting with syslog-ng OSE version 3.2, the `DATE` macro equals the `S_DATE` macro. In earlier versions the value of `DATE` depended on the `use_time_recv()` global option, which was removed from syslog-ng OSE.

**Warning**

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



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

**Note**

Earlier versions of syslog-ng used templates and scripts to send log messages into SQL databases. Starting from version 2.1, syslog-ng natively supports direct database access using the `sql()` destination. For details, see *Section 7.4.4, sql() destination options (p. 102)*.

**Example 11.1. Using templates and macros**

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

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

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

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

The following file destination uses macros to daily create separate logfiles for every client host.

```
destination d_file {
    file("/var/log/$YEAR.$MONTH.$DAY/$HOST.log");
};
```

**Note**

Macros can be used to format messages, and also in the name of destination files or database tables. However, they cannot be used in sources as wildcards, for example, to read messages from files or directories that include a date in their name.

11.1.3. Macros of syslog-ng OSE

The following macros are available in syslog-ng OSE.

BSDTAG

Description: Facility/priority information in the format used by the FreeBSD syslogd: a priority number followed by a letter that indicates the facility. The priority number can range from `0` to `7`. The facility letter can range from `A` to `Y`, where `A` corresponds to facility number zero (LOG_KERN), `B` corresponds to facility 1 (LOG_USER), and so on.

Custom macros

Description: CSV parsers and pattern databases can also define macros from the content of the messages, for example, a pattern database rule can extract the username from a login message and create a macro that references the username. For details on using custom macros created with CSV parsers and pattern databases, see *Chapter 12, Parsing and segmenting structured messages (p. 168)* and *Section 13.2.1, Using parser results in filters and templates (p. 177)*, respectively.

**DATE, R_DATE, S_DATE**

Description: Date of the message using the BSD-syslog style timestamp format (month/day/hour/minute/second, each expressed in two digits). This is the original syslog time stamp without year information, for example: *Jun 13 15:58:00*.

DAY, R_DAY, S_DAY

Description: The day the message was sent.

FACILITY

Description: The name of the facility (for example, *kern*) that sent the message.

FACILITY_NUM

Description: The numerical code of the facility (for example, *0*) that sent the message.

FULLDATE, R_FULLDATE, S_FULLDATE

Description: A nonstandard format for the date of the message using the same format as *DATE*, but including the year as well, for example: *2006 Jun 13 15:58:00*.

FULLHOST

Description: The full FQDN of the host name chain (without trimming chained hosts), including the domain name. To use this macro, make sure that the *keep_hostname()* option is enabled.

FULLHOST_FROM

Description: FQDN of the host that sent the message to syslog-ng as resolved by syslog-ng using DNS. If the message traverses several hosts, this is the last host in the chain. To use this macro, make sure that the *keep_hostname()* option is enabled.

The syslog-ng OSE application uses the following procedure to determine the value of the *\$FULLHOST_FROM* macro:

1. The syslog-ng OSE application takes the IP address of the host sending the message.
2. If the *use_dns()* option is enabled, syslog-ng OSE attempts to resolve the IP address to a hostname. If it succeeds, the returned hostname will be the value of the *\$FULLHOST_FROM* macro.
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.

HOUR, R_HOUR, S_HOUR

Description: The hour of day the message was sent.

HOST

Description: The name of the source host where the message originates from. If the message traverses several hosts and the *chain_hostnames()* option is on, the first host in the chain is used. To use this macro, make sure that the *keep_hostname()* option is enabled.



HOST_FROM

Description: Name of the host that sent the message to syslog-ng, as resolved by syslog-ng using DNS. If the message traverses several hosts, this is the last host in the chain. To use this macro, make sure that the `keep_hostname()` option is enabled.

ISODATE, R_ISODATE, S_ISODATE

Description: Date of the message in the ISO 8601 compatible standard timestamp format (yyyy-mm-ddThh:mm:ss+-ZONE), for example: `2006-06-13T15:58:00.123+01:00`. If possible, it is recommended to use `ISODATE` for timestamping. Note that syslog-ng can produce fractions of a second (for example milliseconds) in the timestamp by using the `frac_digits()` global or per-destination option.

LEVEL_NUM

Description: The priority (also called severity) of the message, represented as a numeric value, for example, 3.

MIN, R_MIN, S_MIN

Description: The minute the message was sent.

MONTH, R_MONTH, S_MONTH

Description: The month the message was sent as a decimal value, prefixed with a zero if smaller than 10.

MONTH_ABBREV, R_MONTH_ABBREV, S_MONTH_ABBREV

Description: The English abbreviation of the month name (3 letters).

MONTH_NAME, R_MONTH_NAME, S_MONTH_NAME

Description: The English name of the month name.

MONTH_WEEK, R_MONTH_WEEK, S_MONTH_WEEK

Description: The number of the week in the given month (0-5). The week with numerical value 1 is the first week containing a Monday. The days of month before the first Monday are considered week 0. For example, if a 31-day month begins on a Sunday, then the 1st of the month is week 0, and the end of the month (the 30th and 31st) is week 5.

MSG or MESSAGE

Description: Text contents of the log message without the program name and pid. Note that this has changed in syslog-ng version 3.0; in earlier versions this macro included the program name and the pid. In syslog-ng 3.0, the `MSG` macro became equivalent with the `MSGONLY` macro. The program name and the pid together are available in the `MSGHDR` macro.

MSGHDR

Description: The name and the pid of the program that sent the log message in `PROGRAM: PID` format. Includes a trailing whitespace. Note that the macro returns an empty value if both the program and pid fields of the message are empty.

MSGONLY

Description: Message contents without the program name or pid.



PID

Description: The PID of the program sending the message.

PRI

Description: The priority and facility encoded as a 2 or 3 digit decimal number as it is present in syslog messages.

PRIORITY or LEVEL

Description: The priority (also called severity) of the message, for example, `error`.

PROGRAM

Description: The name of the program sending the message. Note that the content of the \$PROGRAM variable may not be completely trusted as it is provided by the client program that constructed the message.

SDATA, .SDATA.SDID.SDNAME

Description: The syslog-ng application automatically parses the STRUCTURED-DATA part of IETF-syslog messages, which can be referenced in macros. The `$SDATA` macro references the entire STRUCTURED-DATA part of the message, while structured data elements can be referenced using the `$.SDATA.SDID.SDNAME` macro.



Note

When using STRUCTURED-DATA macros, consider the following:

- When referencing an element of the structured data, the macro must begin with the dot (.) character. For example, `$.SDATA.timeQuality.isSynced`.
- The SDID and SDNAME parts of the macro names are case sensitive: `$.SDATA.timeQuality.isSynced` is not the same as `$.SDATA.TIMEQUALITY.ISSYNCED`.



Example 11.2. Using SDATA macros

For example, if a log message contains the following structured data: `[exampleSDID@0 iut="3" eventSource="Application" eventID="1011"] [examplePriority@0 class="high"]` you can use macros like: `$(.SDATA.exampleSDID@0.eventSource}` — this would return the `Application` string in this case.

SEC, R_SEC, S_SEC

Description: The second the message was sent.

SEQNUM

Description: The sequence number of the message is a unique identifier of the message between the end-points. The syslog-ng client calculates this number when processing a new message from a local source; it is not calculated for relayed messages. The sequence number increases for every message, and is not lost even if syslog-ng is reloaded or restarted. The sequence number is a part of every message that uses the new IETF-syslog protocol (`.SDATA.meta.sequenceId`), and can be added to BSD-syslog messages using this macro.

SOURCEIP

Description: IP address of the host that sent the message to syslog-ng. (That is, the IP address of the host in the `FULLHOST_FROM` macro.) Please note that when a message traverses several relays, this macro contains the IP of the last relay.



STAMP, R_STAMP, S_STAMP

Description: A timestamp formatted according to the `_ts_format()` global or per-destination option.

TAG

Description: The priority and facility encoded as a 2 digit hexadecimal number.

TAGS

Description: A comma-separated list of the tags assigned to the message.



Note

Note that the tags are not part of the log message and are not automatically transferred from a client to the server. For example, if a client uses a pattern database to tag the messages, the tags are not transferred to the server. A way of transferring the tags is to explicitly add them to the log messages using a template and the `$TAGS` macro, or to add them to the structured metadata part of messages when using the IETF-syslog message format.

When sent as structured metadata, it is possible to reference to the list of tags on the central server, and for example, to add them to a database column.

TZ, R_TZ, S_TZ

Description: Equivalent to TZOFFSET, used to mean the time zone name abbreviation in syslog-ng 1.6.x.

TZOFFSET, R_TZOFFSET, S_TZOFFSET

Description: The time-zone as hour offset from GMT; for example: `-07:00`. In syslog-ng 1.6.x this used to be `-0700` but as `ISODATE` requires the colon it was added to `TZOFFSET` as well.

UNIXTIME, R_UNIXTIME, S_UNIXTIME

Description: Standard unix timestamp, represented as the number of seconds since `1970-01-01T00:00:00`.

YEAR, R_YEAR, S_YEAR

Description: The year the message was sent.

WEEK, R_WEEK, S_WEEK

Description: The week number of the year, prefixed with a zero for the first nine week of the year. (The first Monday in the year marks the first week.)

WEEK_ABBREV, R_WEEK_ABBREV, S_WEEK_ABBREV

Description: The 3-letter English abbreviation of the name of the day the message was sent, for example `Thu`.

WEEK_DAY, R_WEEK_DAY, S_WEEK_DAY

Description: The day of the week as a numerical value (1-7).

WEEKDAY, R_WEEKDAY, S_WEEKDAY

Description: These macros are deprecated, use `WEEK_ABBREV, R_WEEK_ABBREV, S_WEEK_ABBREV` instead. The 3-letter name of the day of week the message was sent, for example `Thu`.



WEEK_DAY_NAME, R_WEEK_DAY_NAME, S_WEEK_DAY_NAME

Description: The English name of the day.

11.1.4. Using template functions

A template function is a transformation: it modifies the way macros or name-value pairs are expanded. Template functions can be used in template definitions, or when macros are used in the configuration of syslog-*ng* OSE. Template functions use the following syntax:

```
$ (function-name parameter1 parameter2 parameter3 ...)
```

For example, the `$ (echo)` template function simply returns the value of the macro it receives as a parameter, thus `$ (echo $HOST)` is equivalent to `$HOST`

The parameters of template functions are separated by a whitespace character. If you want to use a longer string or multiple macros as a single parameter, enclose the parameter in double-quotes or apostrophes. For example:

```
$ (echo "$HOST $PROGRAM $PID")
```

Template functions can be nested into each other, so the parameter of a template function can be another template function, like:

```
$ (echo $ (echo $HOST))
```

For details on using template functions, see the descriptions of the individual template functions in *Section 11.1.5, Template functions of syslog-*ng* OSE (p. 160)*.

11.1.5. Template functions of syslog-*ng* OSE

The following template functions are available in syslog-*ng* OSE.

echo

Syntax:

```
$ (echo argument)
```

Description: Returns the value of its argument. using `$ (echo $HOST)` is equivalent to `$HOST`.

grep

Syntax:

```
$ (grep condition value-to-select)
```

Description: The `grep` template function is useful when using a pattern database to correlate related log messages. The `grep` template function can be used to filter the messages of the same context when the index of the particular message is not known.

**Example 11.3. Using the grep template function**

The following example selects the message of the context that has a `username` name-value pair with the `root` value, and returns the value of the `auth_method` name-value pair.

```
$(grep ("${username}" == "root") ${auth_method})
```

It is possible to specify multiple name-value pairs as parameters, separated with commas. If multiple messages match the condition of `grep`, these will be returned also separated by commas. This can be used for example to collect the e-mail recipients from postfix messages.

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 11.4. Using pattern databases and the if template function**

The following example returns `violation` if the `username` name-value pair of a message processed with pattern database is `root`, and `system` otherwise.

```
$(if ("${username}" == "root") "violation" "system")
```

This can be used to set the class of a message in pattern database rules based on the condition.

```
<value name="username">$(if ("${username}" == "root") "violation" "system")</value>
```

Since template functions can be embedded into each other, it is possible to use another template function as the template of the first one. For example, the following expression returns `root` if the username is `root`, `admin` if the username is `joe`, and `normal user` otherwise.

```
<value name="username">
  $(if ("${username}" == "root")
    "root"
    $(if ("${username}" == "joe") "admin" "normal user")
    "normal user")</value>
```

ipv4-to-int**Syntax:**

```
$(ipv4-to-int parameter)
```

Description: Converts the specified IPv4 address to its numeric representation. The numerical value of an IPv4 address is calculated by treating the IP address as a 4-byte hexadecimal value. For example, the 192.168.1.1 address equals to: 192=C0, 168=A8, 1=01, 1=01, or C0A80101, which is 3232235777 in decimal representation.

**Note**

This template function is available only if the `convertfuncs` module has been loaded. By default, syslog-ng OSE loads every available module.



11.2. Modifying messages

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

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

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

Rewriting messages is often used in conjunction with message parsing *Section 12.1, Parsing messages (p. 168)*.

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

**Note**

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

To create replace a part of the log message, define the string or regular expression to replace, the string to replace the original text (macros can be used as well), and the field of the message that the rewrite rule should process. Substitution rules can operate on any value available via macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (for details, see *Chapter 12, Parsing and segmenting structured messages (p. 168)* and *Chapter 13, Processing message content with a pattern database (p. 173)*). The only exceptions are the FACILITY, SEVERITY, TAGS, and the date-related fields, which cannot be rewritten. As of syslog-ng 3.1, it is also possible to rewrite the structured-data fields of messages complying to the RFC5424 (IETF-syslog) message format. Substitution rules use the following syntax:

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

The `type()` and `flags()` options are optional. The `type()` specifies the type of regular expression to use; while the `flags()` are the flags of the regular expressions. For details on regular expressions, see *Section 11.3, Regular expressions (p. 164)*.

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

**Tip**

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

**Example 11.5. Using substitution rules**

The following example replaces the first occurrence of the string `IP` in the text of the message with the string `IP-Address`.

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

To replace every occurrence, use:

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

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

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

To set a field of the message to a specific value, define the string to include in the message, and the field where it should be included. Setting a field can operate on any value available via macros, for example HOST, MESSAGE, PROGRAM, or any user-defined macros created using parsers (for details, see *Chapter 12, Parsing and segmenting structured messages (p. 168)* and *Chapter 13, Processing message content with a pattern database (p. 173)*). The only exceptions are the FACILITY, SEVERITY, TAGS, and the date-related fields, which cannot be rewritten. Note that the rewrite operation completely replaces any previous value of that field. Use the following syntax:

Declaration:

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

**Example 11.6. Setting message fields to a particular value**

The following example sets the HOST field of the message to `myhost`.

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

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

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

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

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

11.2.1. Conditional rewrites

Starting with version 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.

**Tip**

Using conditions in rewrite rules can simplify your syslog-ng OSE configuration file, as you do not need to create separate log paths to modify certain messages.

**Example 11.7. 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)));};
```

11.3. 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) " (quotation mark) must be escaped (\ \ and \ "). The following expressions are interpreted: \a; \n; \r; \t; \v. For example, the *\\$40* expression matches the *\$40* string. Backslashes have to be escaped as well if they are meant literally, for example, the *\ \d* expression matches the *\d* string.

**Tip**

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

- Enclosing normal strings between double-quotes ("*string*") is not necessary, you can just omit the double-quotes. For example when writing filters, *match ("sometext")* and *match(sometext)* will both match for the *sometext* string.

**Note**

Only strings without whitespace or special characters can be used without quotes or double quotes.



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

The regular expressions can use up to 255 regexp matches (`$(1) ... $(255)`), but only from the last filter and only if the `flags("store-matches")` flag was set for the filter. For case-insensitive searches, use the `flags("ignore-case")` option.

11.3.1. Types and options of regular expressions

By default, syslog-ng uses POSIX-style regular expressions. To use other expression types, add the `type()` option after the regular expression.

The syslog-ng OSE application supports the following expression types:

- [POSIX regular expressions](#)
- [Perl Compatible Regular Expressions \(PCRE\)](#)
- [Literal string searches](#)
- [Glob patterns without regular expression support](#)

posix

Description: Use POSIX regular expressions. If the `type()` parameter is not specified, syslog-ng uses POSIX regular expressions by default.

Posix regular expressions have the following flag options:

global: Usable only in rewrite rules; match for every occurrence of the expression, not only the first one.

ignore-case: Disable case-sensitivity.

store-matches: Store the matches of the regular expression into the `$1, ... $255` variables. Matches from the last filter expression can be referenced in regular expressions.

utf8: Use UTF-8 matching.



Example 11.8. Using Posix regular expressions

```
filter f_message { message("keyword" flags("utf8" "ignore-case")); }
```

pcre

Description: Use Perl Compatible Regular Expressions (PCRE). PCRE expressions can be used if syslog-ng OSE was compiled with the `--enable-pcre` option enabled. Execute the `syslog-ng -V` command to check if your binary supports PCRE regular expressions. Starting with syslog-ng OSE version 3.1, PCRE expressions are supported on every platform.

PCRE regular expressions have the following flag options:



global: Usable only in rewrite rules; match for every occurrence of the expression, not only the first one.

ignore-case: Disable case-sensitivity.

nobackref: Do not store back references for the matches — improves performance.

store-matches: Store the matches of the regular expression into the `$1`, ... `$255` variables. Named matches (also called named subpatterns), for example `(?<name>...)`, are stored as well. Matches from the last filter expression can be referenced in regular expressions.

unicode: Use Unicode support for UTF-8 matches: UTF-8 character sequences are handled as single characters.

utf8: An alias for the `unicode` flag.



Example 11.9. Using PCRE regular expressions

```
rewrite r_rewrite_subst
    {subst("a*", "?", field("message") type("pcre") flags("utf8" "global")); };
```

string

Description: Match the strings literally, without regular expression support. By default, only identical strings are matched. For partial matches, use the `flags ("prefix")` or the `flags ("substring")` flags.

glob

Description: Use glob patterns (that is, wildcards and character ranges) without regular expression support. The advantage of glob patterns to regular expressions is that globs can be processed much faster. For details on glob patterns, see the *glob manual page* (`man glob`).

11.3.2. Optimizing regular expressions

The `host()`, `match()`, and `program()` filter functions and some other syslog-*ng* objects accept regular expressions as parameters. But evaluating general regular expressions puts a high load on the CPU, which can cause problems when the message traffic is very high. Often the regular expression can be replaced with simple filter functions and logical operators. Using simple filters and logical operators, the same effect can be achieved at a much lower CPU load.



Example 11.10. Optimizing regular expressions in filters

Suppose you need a filter that matches the following error message logged by the `xntp` NTP daemon:

```
xntp[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"); };
```



Chapter 12. Parsing and segmenting structured messages

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

For details on using parsers, see *Section 12.1, Parsing messages (p. 168)* and *Section 12.2, Options of CSV parsers (p. 170)*.

12.1. Parsing messages

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

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



Note

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

To create a parser, define the columns of the message, the delimiter or separator characters, and optionally the characters that are used to escape the delimiter characters (quote-pairs). For the list of parser parameters, see *Section 12.2, Options of CSV parsers (p. 170)*.

Declaration:

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

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



Example 12.1. Segmenting hostnames separated with a dash

The following example separates hostnames like `example-1` and `example-2` into two parts.

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



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



Example 12.2. Parsing Apache log files

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

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

Here is a sample message:

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

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

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

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

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

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

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



Example 12.3. Segmenting a part of a message

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

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

log { source(s_local);
      log { parser(p_apache); parser(p_apache_timestamp); destination(d_file); };
};
```

Further examples:

- For an example on using the `greedy` option, see *Example 12.4, Adding the end of the message to the last column (p. 171)*.



12.2. Options of CSV parsers

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

To create a parser, define the columns of the message, the delimiter or separator characters, and optionally the characters that are used to escape the delimiter characters (quote-pairs).

Declaration:

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

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

csv-parser

Synopsis: `csv-parser(columns("PARSER.COLUMN1", "PARSER.COLUMN2", ...))`

Description: Specifies the type of parser to use, and the name of the columns to separate messages to. Currently only the `csv-parser` is implemented, which can separate columns based on delimiter characters and strings.

delimiters

Synopsis: `delimiters("<delimiter_characters>")`

Description: The character that separates the columns in the message.

flags()

Synopsis: `drop-invalid, escape-none, escape-backslash, escape-double-char, greedy, strip-whitespace`

Description: Specifies various options for parsing the message. The following flags are available:

- *drop-invalid*: When the `drop-invalid` option is set, the parser does not process messages that do not match the parser. For example, a message does not match the parser if it has less columns than specified in the parser, or it has more columns but the `greedy` flag is not enabled. Using the `drop-invalid` option practically turns the parser into a special filter, that matches messages that have the predefined number of columns (using the specified delimiters).



Tip

Messages dropped as invalid can be processed by a `fallback` log path. For details on the `fallback` option, see *Section 8.1.2, Log path flags (p. 124)*.



- *escape-backslash*: The parsed message uses the backslash (\) character to escape quote characters.
- *escape-double-char*: The parsed message repeats the quote character when the quote character is used literally. For example, to escape a comma (,), the message contains two commas (,,).
- *escape-none*: The parsed message does not use any escaping for using the quote character literally.
- *greedy*: The *greedy* option assigns the remainder of the message to the last column, regardless of the delimiter characters set. You can use this option to process messages where the number of columns varies.

**Example 12.4. Adding the end of the message to the last column**

If the *greedy* option is enabled, the syslog-ng application adds the not-yet-parsed part of the message to the last column, ignoring any delimiter characters that may appear in this part of the message.

For example, you receive the following comma-separated message: `example 1, example2, example3,` and you segment it with the following parser:

```
csv_parser(columns("COLUMN1", "COLUMN2", "COLUMN3") delimiters(",") );
```

The *COLUMN1*, *COLUMN2*, and *COLUMN3* variables will contain the strings `example1`, `example2`, and `example3`, respectively. If the message looks like `example 1, example2, example3, some more information`, then any text appearing after the third comma (that is, `some more information`) is not parsed, and possibly lost if you use only the variables to reconstruct the message (for example, to send it to different columns of an SQL table).

Using the *greedy* flag will assign the remainder of the message to the last column, so that the *COLUMN1*, *COLUMN2*, and *COLUMN3* variables will contain the strings `example1`, `example2`, and `example3, some more information`.

```
csv_parser(columns("COLUMN1", "COLUMN2", "COLUMN3") delimiters(",") flags(greedy));
```

- *strip-whitespace*: The *strip-whitespace* flag removes trailing whitespaces from the beginning and the end of the columns.

quote-pairs()

Synopsis: `quote-pairs('<quote_pairs>')`

Description: List quote-pairs between single quotes. Delimiter characters enclosed between quote characters are ignored. Note that the beginning and ending quote character does not have to be identical, for example `[]` can also be a quote-pair. For an example of using *quote-pairs()* to parse Apache log files, see *Example 12.2, Parsing Apache log files* (p. 169).

template()

Synopsis: `template("${{<macroname>}}")`

Description: The macro that contains the part of the message that the parser will process. It can also be a macro created by a previous parser of the log path. By default, this is empty and the parser processes the entire message.



For examples, see *Example 12.1, Segmenting hostnames separated with a dash* (p. 168) and *Example 12.3, Segmenting a part of a message* (p. 169).



Chapter 13. Processing message content with a pattern database

13.1. Classifying log messages

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

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

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

To make the message classification more flexible and robust, the patterns can contain pattern parsers: elements that match on a set of characters. For example, the NUMBER parser matches on any integer or hexadecimal number (for example 1, 123, 894054, 0xFFFF, and so on). Other pattern parsers match on various strings and IP addresses. For the details of available pattern parsers, see *Section 13.5.1, Using pattern parsers (p. 182)*.

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

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

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

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

For details on using pattern databases to classify log messages, see *Section 13.2, Using pattern databases (p. 176)*.



13.1.1. The structure of the pattern database

The pattern database is organized as follows:

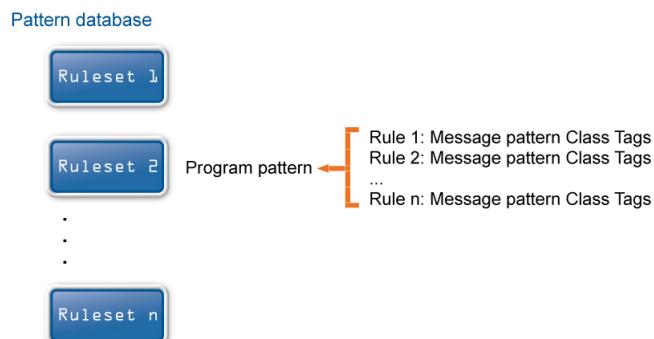


Figure 13.1. The structure of the pattern database

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



Note

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

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



13.1.2. How pattern matching works



Figure 13.2. Applying patterns

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

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



Note

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

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

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

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 `Appo`t and a pattern parser `Ap@STRING@`, the `Appo`t pattern is matched. If the literal does not match the incoming string (for example, `Apple`), syslog-ng attempts to match the pattern with the parser. However, if there are two or more parsers on the same level, only the first one will be applied, even if it does not perfectly match the message.

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

13.1.3. Artificial ignorance

Artificial ignorance is a method to detect anomalies. When applied to log analysis, it means that you ignore the regular, common log messages - these are the result of the regular behavior of your system, and therefore are not too interesting. However, new messages that have not appeared in the logs before can sign important events, and



should be therefore investigated. "By definition, something we have never seen before is anomalous" (Marcus J. Ranum).

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

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

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

13.2. Using pattern databases

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

Declaration:

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

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



Example 13.1. Defining pattern databases

The following statement uses the database located at `/opt/syslog-ng/var/db/patterndb.xml`.

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

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

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



Note

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

**Example 13.2. Using classification results**

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

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

template("${.classifier.rule_id};${_UNIXTIME};${SOURCEIP};${HOST};${PROGRAM};${PID};${MSG}\n")
    template_escape(no)
);
};
```

For details on how to create your own pattern databases see *Section 13.5.3, The syslog-ng pattern database format (p. 184)*.

13.2.1. Using parser results in filters and templates

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

**Example 13.3. Using classification results for filtering messages**

To filter on a specific message class, create a filter that checks the `.classifier.class` macro, and use this filter in a log statement.

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

log {
    source(s_all);
    parser(pattern_db);
    filter(fi_classViolation);
    destination(di_classViolation);
};
```

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_classRule {
    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**

Starting with version 3.2, syslog-ng OSE 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 13.4. Using pattern parsers as macros**

For example, you want to parse messages of an application that look like "`Transaction: <type>`.", where `<type>` is a string that has different values (for example refused, accepted, incomplete, and so on). To parse these messages, you can use the following pattern:

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

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

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

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

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

**Note**

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

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

13.2.2. Downloading sample pattern databases

Sample pattern databases are available at the [BalaBit Download page](#). Note that even though these pattern databases contain over 8000 rules for more than 200 applications and devices, they are only samples and experimental databases that are not officially supported and may or may not work in your environment.

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

- you are free to use and modify the patterns purposes;
- when redistributing the pattern databases you must distribute your modifications under the same license;
- and when redistributing the pattern databases, you must make it obvious that the original syslog-ng pattern databases are [available here](#).

For legal details, the full text of the license is [available here](#).



13.3. Correlating log messages

Starting with version 3.2, the syslog-ng OSE application is able to correlate log messages identified using pattern databases.

Log messages are supposed to describe events, but applications often separate information about a single event into different log messages. For example, the Postfix e-mail server logs the sender and recipient addresses into separate log messages, or in case of an unsuccessful login attempt, the OpenSSH server sends a log message about the authentication failure, and the reason of the failure in the next message.

Of course, messages that are not so directly related can be correlated as well, for example, login-logout messages, and so on.

To correlate log messages, syslog-ng OSE uses the pattern database to add messages into message-groups called contexts. A context consists of a series of log messages that are related to each other in some way, for example, the log messages of an SSH session can belong to the same context. As new messages come in, they may be added to a context. Also, when an incoming message is identified it can trigger actions to be performed, for example, generate a new message that contains all the important information that was stored previously in the context. (For details on triggering actions and generating messages, see *Section 13.4, “Triggering actions for identified messages”* (p. 180).)

There are two attributes for pattern database rules that determine if a message matching the rule is added to a context: *context-scope* and *context-id*. The *context-scope* attribute acts as an early filter, selecting messages sent by the same process (\$HOST\$PROGRAM\$PID is identical), application (\$HOST\$PROGRAM is identical), or host, while the *context-id* actually adds the message to the context specified in the id. The *context-id* can be a simple string, or can contain macros or values extracted from the log messages for further filtering.

**Note**

Message contexts are persistent and are not lost even if syslog-ng OSE is restarted.

Another parameter of a rule is the *context-timeout* attribute, which determines how long a context is stored, that is, how long syslog-ng OSE waits for related messages to arrive. Note the following points about timeout values:

- When a new message is added to a context, syslog-ng OSE will restart the timeout using the *context-timeout* set for the new message.
- When calculating if the timeout has already expired or not, syslog-ng OSE uses the timestamps of the incoming messages, not system time elapsed between receiving the two messages (unless the messages do not include a timestamp, or the *keep-timestamp (no)* option is set). That way syslog-ng OSE can be used to process and correlate already existing log messages offline. However, the timestamps of the messages must be in chronological order (that is, a new message cannot be older than the one already processed), and if a message is newer than the current system time (that is, it seems to be coming from the future), syslog-ng OSE will replace its timestamp with the current system time.



Example 13.5. How syslog-ng OSE calculates `context-timeout`

Consider the following two messages:

```
<38>1990-01-01T14:45:25 customhostname program6[1234]: program6 testmessage
<38>1990-01-01T14:46:25 customhostname program6[1234]: program6 testmessage
```

If the `context-timeout` is 10 seconds and syslog-ng OSE receives the messages within 1 sec, the timeout event will occur immediately, because the difference of the two timestamp (60 sec) is larger than the timeout value (10 sec).

- Avoid using unnecessarily long timeout values on high-traffic systems, as storing the contexts for many messages can require considerable memory. For example, if two related messages usually arrive within seconds, it is not needed to set the timeout to several hours.



Example 13.6. Using message correlation

```
<rule id="..." context-id="ssh-session" context-timeout="86400" context-scope="process">
  <patterns>
    <pattern>Accepted @ESTRING:usracct.authmethod: @for @ESTRING:usracct.username:
@from @ESTRING:usracct.device: @port @ESTRING:: @@ANYSTRING:usracct.service@</pattern>
  </patterns>
...
</rule>
```

For details on configuring message correlation, see the description of the `context-id`, `context-timeout`, and `context-scope` attributes of pattern database rules.

13.3.1. 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 `$HOST@1` expression refers to the host field of the previous (second) message in the context, `$PID@2` to the PID of the first message, and so on.

13.4. Triggering actions for identified messages

Starting with version 3.2, the syslog-ng OSE application is able to generate (trigger) messages automatically if certain events occur, for example, a specific log message is received, or the correlation timeout of a message expires. Basically, you can define messages for every pattern database rule that are emitted when a message matching the rule is received. Triggering messages is often used together with message correlation, but can also be used separately.



Note

Currently syslog-ng OSE sends the generated messages to its `internal()` source, but this will change in future versions.



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 13.7. 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 limit when a message is triggered, use the *condition* attribute and specify a filter expression: the action will be executed only if the condition is met. For example, the following action is executed only if the message was sent by the host called *myhost*.

```
<action condition="`${HOST}` == 'example'">
```

For details on configuring actions, see the description of the *pattern database format*.

To perform an external action when a message is triggered, for example, to send the message in an e-mail, you have to route the generated messages to an external application using the *program()* destination.



Example 13.8. Sending triggered messages to external applications

The following sample configuration selects the triggered messages and sends them to an external script.

- 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>
```

- Create a destination that will process the triggered messages.

```
destination d_triggers { program("/bin/myscript"); };
```

- Create a filter that selects the triggered messages from the internal source.

```
filter f_triggers {match("yes" value ("TRIGGER") type(string));};
```

- 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); };
```

- 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
}
```

Certain features of generating messages can be used only if message correlation is used as well.



- The syslog-*ng* OSE application automatically fills the fields for the generated message based on the scope of the context, for example, the HOST and program fields if the *context-scope* is *program*.
- When used together with message correlation, you can also refer to fields and values of earlier messages of the context by adding the `@<distance-of-referenced-message-from-the-current>` suffix to the macro. For details, see *Section 13.3.1, Referencing earlier messages of the context* (p. 180).
- 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">
```

For details on correlating messages, see *Section 13.3, Correlating log messages* (p. 179).

13.5. Creating pattern databases

13.5.1. 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;
- a closing @ character.



Example 13.9. Pattern parser syntax

A simple parser:

```
@STRING@
```

A named parser:

```
@STRING:myparser_name@
```

A named parser with a parameter:

```
@STRING:myparser_name:*@
```

A parser with a parameter, but without a name:

```
@STRING::*:*
```

The following parsers are available:

- `@ANYSTRING@`: Parses everything to the end of the message; you can use it to collect everything that is not parsed specifically to a single macro. In that sense its behavior is similar to the `greedy()` option of the CSV parser.
- `@DOUBLE@`: An obsolete alias of the `@FLOAT@` parser.



- **@ESTRING@:** This parser has a required parameter that acts as the stopcharacter: the parser parses everything until it finds the stopcharacter. For example to stop by the next " (double quote) character, use `@ESTRING:::"@`. To stop by a colon (:), the colon has to be escaped with another colon, like: `@ESTRING:::::@`. As of syslog-*ng* 3.1, it is possible to specify a stopstring instead of a single character, for example, `@ESTRING::stop_here@`. The @ character cannot be a stopcharacter, nor can line-breaks or tabs.
- **@FLOAT@:** A floating-point number that may contain a dot (.) character. (Up to syslog-*ng* 3.1, the name of this parser was `@DOUBLE@`.)
- **@IPv4@:** Parses an IPv4 IP address (numbers separated with a maximum of 3 dots).
- **@IPv6@:** Parses any valid IPv6 IP address.
- **@IPvANY@:** Parses any IP address.
- **@NUMBER@:** A sequence of decimal (0-9) numbers (for example, 1, 0687, and so on). Note that if the number starts with the 0x characters, it is parsed as a hexadecimal number, but only if at least one valid character follows 0x.
- **@QSTRING@:** Parse a string between the quote characters specified as parameter. Note that the quote character can be different at the beginning and the end of the quote, for example: `@QSTRING:::"@` parses everything between two quotation marks ("), while `@QSTRING:<>@` parses from an opening bracket to the closing bracket. The @ character cannot be a quote character, nor can line-breaks or tabs.
- **@STRING@:** A sequence of alphanumeric characters (0-9, A-z), not including any whitespace. Optionally, other accepted characters can be listed as parameters (for example, to parse a complete sentence, add the whitespace as parameter, like: `@STRING::: @`). Note that the @ character cannot be a parameter, nor can line-breaks or tabs.

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 13.10. Using the STRING and ESTRING parsers

For example, if the message is `user=joe96 group=somegroup`, `@STRING:mytext:@` parses only to the first non-alphanumeric character (=), parsing only `user`. `@STRING:mytext:=@` parses the equation mark as well, and proceeds to the next non-alphanumeric character (the whitespace), resulting in `user=joe96` being parsed. `@STRING:mytext:= @` will parse the whitespace as well, and proceed to the next non-alphanumeric non-equation mark non-whitespace character, resulting in `user=joe96 group=somegroup`.

Of course, usually it is better to parse the different values separately, like this: `"user=@STRING:user@group=@STRING:group@"`.

If the username or the group may contain non-alphanumeric characters, you can either include these in the second parameter of the parser (as shown at the beginning of this example), or use an ESTRING parser to parse the message till the next whitespace: `"user=@ESTRING:user: @group=@ESTRING:group: @"`.

13.5.2. What's new in the syslog-*ng* pattern database format V4

The V4 database format has the following differences compared to the V3 format:



- It is now possible to specify multiple program patterns for a ruleset. For details, see the description of the [*patterns tag*](#).
- The <value> element of name-value pairs can include template functions. For details, see *Section 11.1.4, Using template functions (p. 160)*, for examples, see *Section if (p. 161)*.
- It is now possible to correlate log messages processed with the pattern database. For details, see *Section 13.3, Correlating log messages (p. 179)*.
- It is now possible to generate new messages based on pattern matching and correlation results. For details, see *Section 13.4, Triggering actions for identified messages (p. 180)* and the description of the [*actions tag*](#).

13.5.3. The syslog-ng pattern database format

Pattern databases are XML files that contain rules describing the message patterns. For sample pattern databases, see *Section 13.2.2, Downloading sample pattern databases (p. 178)*.

The following scheme describes the V4 format of the pattern database. This format is used by syslog-ng OSE 3.2 and later, and is backwards-compatible with the earlier V3 format.

For a sample database containing only a single pattern, see *Example 13.11, A V4 pattern database containing a single rule (p. 189)*.



Tip

Use the `pdbscript` utility that is bundled with syslog-ng to test message patterns and convert existing databases to the latest format. For details, see *pdbscript(1) (p. 209)*.

To automatically create an initial pattern database from an existing log file, use the `pdbscript patternize` command. For details, see the section called “*The patternize command*” (p. 211).

- **<patterndb>**: The container element of the pattern database. For example:

```
<patterndb version='4' pub_date='2010-10-25'>
```

- *version*: The schema version of the pattern database. The current version is 4.
- *pubdate*: The publication date of the XML file.
- **<ruleset>**: A container element to group log patterns for an application or program. For example:

```
<ruleset name='su' id='480de478-d4a6-4a7f-bea4-0c0245d361e1'>
```

A `<patterndb>` element may contain any number of `<ruleset>` elements.

- *name*: The name of the application. Note that the function of this attribute is to make the database more readable, syslog-ng uses the `<pattern>` element to identify the applications sending log messages.
- *id*: A unique ID of the application, for example, the md5 sum of the *name* attribute.
- **description**: OPTIONAL — A description of the ruleset or the application.



- **url:** OPTIONAL — An URL referring to further information about the ruleset or the application.
- **<patterns>:** A container element storing program names also called *program pattern*. For example:

```
<patterns>
  <pattern>su</pattern>
<patterns>
```

A *<patterns>* element may contain any number of **<pattern>** elements.

- **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.

```
<patterns>
  <pattern>firstapplication</pattern>
  <pattern>otherapplication</pattern>
<patterns>
```

It is not necessary to use multiple patterns if only the end of the \$PROGRAM fields is different, use only the beginning of the \$PROGRAM field as the *pattern*. For example, the Postfix e-mail server sends messages using different process names, but all of them begin with the *postfix* string.



Note

If the *<pattern>* element of a ruleset is not specified, syslog-*ng* OSE will use this ruleset as a fallback ruleset: it will apply the ruleset to messages that have an empty PROGRAM header, or if none of the program patterns matched the PROGRAM header of the incoming message.

- **<rules>:** A container element for the rules of the ruleset.
- **<rule>:** An element containing message patterns and how a message that matches these patterns is classified. For example:

```
<rule provider='balabit' id='f57196aa-75fd-11dd-9bba-001e6806451b'
  class='violation'>
```

The following example specifies attributes for correlating messages as well. For details on correlating messages, see *Section 13.3, Correlating log messages (p. 179)*.

```
<rule provider='balabit' id='f57196aa-75fd-11dd-9bba-001e6806451b'
  class='violation' context-id='same-session' context-scope='process'
  context-timeout='360'>
```



Note

If the following characters appear in the message, they must be escaped in the rule as follows:

- @: Use @@, for example user@@example.com
- <: Use <



- `>`: Use `>`
- `&`: Use `&`

The `<rules>` element may contain any number of `<rule>` elements.

- *provider*: The provider of the rule. This is used to distinguish between who supplied the rule; that is, if it has been created by BalaBit, or added to the xml by a local user.
- *id*: The globally unique ID of the rule.
- *class*: The class of the rule — syslog-*ng* assigns this class to the messages matching a pattern of this rule.
- *context-id*: OPTIONAL — An identifier to group related log messages when using the pattern database to correlate events. The ID can be a descriptive string describing the events related to the log message (for example, `ssh-sessions` for log messages related to SSH traffic), but can also contain macros to generate IDs dynamically. When using macros in IDs, see also the `context-scope` attribute. For details on correlating messages, see *Section 13.3, Correlating log messages (p. 179)*.



Note

The syslog-*ng* OSE application determines the context of the message *after* the pattern matching is completed. This means that macros and name-value pairs created by the matching pattern database rule can be used as `context-id` macros.

- *context-timeout*: OPTIONAL — The number of seconds the context is stored. Note that for high-traffic logservers, storing open contexts for long time can require significant amount of memory. For details on correlating messages, see *Section 13.3, Correlating log messages (p. 179)*.
- *context-scope*: OPTIONAL — Specifies which messages belong to the same context. This attribute is used to determine the context of the message if the `context-id` does not specify any macros. Usually, `context-scope` acts a filter for the context, with `context-id` refining the filtering if needed. The `context-scope` attribute has the following possible values:
 - *process*: Only messages that are generated by the same process of a client belong to the same context, that is, messages that have identical `$HOST`, `$PROGRAM` and `$PID` values. This is the default behavior of syslog-*ng* OSE if `context-scope` is not specified.
 - *program*: Messages that are generated by the same application of a client belong to the same context, that is, messages that have identical `$HOST` and `$PROGRAM` values.
 - *host*: Every message generated by a client belongs to the same context, only the `$HOST` value of the messages must be identical.
 - *global*: Every message belongs to the same context.

**Note**

Using the `context-scope` attribute is significantly faster than using macros in the `context-id` attribute.

For details on correlating messages, see *Section 13.3, Correlating log messages (p. 179)*.

- **<patterns>**: An element containing the patterns of the rule. If a **<patterns>** element contains multiple **<pattern>** elements, the class of the **<rule>** is assigned to every syslog message matching any of the patterns.
 - **<pattern>**: A pattern describing a log message. This element is also called *message pattern*. For example:
- ```
<pattern>+ ??? root-</pattern>
```
- **description**: OPTIONAL — A description of the pattern or the log message matching the pattern.
  - **urls**: OPTIONAL — An element containing one or more URLs referring to further information about the patterns or the matching log messages.
    - **url**: OPTIONAL — An URL referring to further information about the patterns or the matching log messages.
  - **values**: OPTIONAL — Name-value pairs that are assigned to messages matching the patterns, for example, the representation of the event 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.
    - **value**: OPTIONAL — Contains the value of the name-value pair that is assigned to the message. For example:

```
<value name=".classifier.outcome">/Success</value>
```

The **<value>** element of name-value pairs can include template functions. For details, see *Section 11.1.4, Using template functions (p. 160)*, for examples, see *Section if (p. 161)*.

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 *Section 13.3, Correlating log messages (p. 179)*.

- **name**: The name of the name-value pair. It can also be used as a macro to reference the assigned value.
- **examples**: OPTIONAL — A container element for sample log messages that should be recognized by the pattern. These messages can be used also to test the patterns and the parsers.
- **example**: OPTIONAL — A container element for a sample log message.



- **test\_message:** OPTIONAL — A sample log message that should match this pattern. For example:

```
<test_message program="myapplication">Content filter has been
enabled</test_message>
```

- *program*: The program pattern of the test message. For example:

```
<test_message program="proftpd">ubuntu
(:ffff:192.168.2.179[::ffff:192.168.2.179]) - FTP session
closed.</test_message>
```

- **test\_values:** OPTIONAL — A container element to test the results of the parsers used in the pattern.

- **test\_value:** OPTIONAL — The expected value of the parser when matching the pattern to the test message. For example:

```
<test_value name=".dict.ContentFilter">enabled</test_value>
```

- *name*: The name of the parser to test.

- **actions:** OPTIONAL — A container element for actions that are performed if a message is recognized by the pattern. For details on actions, see *Section 13.4, Triggering actions for identified messages (p. 180)*.

- **action:** OPTIONAL — A container element describing an action that is performed when a message matching the rule is received.

- *condition*: A syslog-*ng* filter expression. The action is performed only if the message matches the filter. The filter can include macros and name-value pairs extracted from the message.

- *rate*: Specifies maximum how many messages should be generated in the specified time period in the following format: <number-of-messages>/<period-in-seconds>. For example: *1/60* allows 1 message per minute. Rates apply within the scope of the context, that is, if *context-scope="host"* and *rate="1/60"*, then maximum one message is generated per minute for every host that sends a log message matching the rule. Excess messages are dropped. Note that when applying the rate to the generated messages, syslog-*ng* OSE uses the timestamps of the log messages, similarly to calculating the *context-timeout*. That way *rate* is applied correctly even if the log messages are processed offline.

- *trigger*: Specifies when the action is executed. The *trigger* attribute has the following possible values:

- *match*: Execute the action immediately when a message matching the rule is received.

- *timeout*: Execute the action when the correlation timer (*context-timeout*) expires. This is available only if actions are used together with correlating messages.



- **message:** A container element storing the message to be sent when the action is executed. Currently syslog-ng OSE sends these messages to the *internal()* destination.
- **values:** A container element for values and fields that are used to create the message generated by the action.
  - **value:** Sets the value of the message field specified in the *name* attribute of the element. For example, to specify the body of the generated message, use the following syntax:

```
<value name="MESSAGE">A log message matched rule number
$.classifier.rule_id</value>
```

Note that currently it is not possible to add DATE, FACILITY, or SEVERITY fields to the message.

When the action is used together with message correlation, the syslog-ng OSE application automatically adds fields to the message based on the *context-scope* parameter. For example, using *context-scope="process"* automatically fills the HOST, PROGRAM, and PID fields of the generated message.

- **name:** Name of the message field set by the *value* element.

- **tags:** OPTIONAL — An element containing custom keywords (tags) about the messages matching the patterns. The tags can be used to label specific events (for example user logons). It is also possible to filter on these tags later (for details, see *Section 8.3.5, Tagging messages (p. 132)*). Starting with syslog-ng Open Source Edition 3.2, the list of tags assigned to a message can be referenced with the *\$TAGS* macro.

- **tag:** OPTIONAL — A keyword or tags applied to messages matching the rule. For example:

```
<tags><tag>UserLogin</tag></tags>
```



#### Example 13.11. A V4 pattern database containing a single rule

The following pattern database contains a single rule that matches a log message of the *ssh* application. A sample log message looks like:

```
Accepted password for sampleuser from 10.50.0.247 port 42156 ssh2
```

The following is a simple pattern database containing a matching rule.

```
<patterndb version='4' pub_date='2010-10-17'>
 <ruleset name='ssh' id='123456678'>
 <pattern>ssh</pattern>
 <rules>
 <rule provider='me' id='182437592347598' class='system'>
 <patterns>
 <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @_for@QSTRING:SSH_USERNAME: @_from\ @_QSTRING:SSH_CLIENT_ADDRESS: @_port @_NUMBER:SSH_PORT_NUMBER:@ssh2</pattern>
 </patterns>
 </rule>
 </rules>
 </ruleset>
</patterndb>
```

Note that the rule uses macros that refer to parts of the message, for example, you can use the *\$SSH\_USERNAME* macro refer to the username used in the connection.



The following is the same example, but with a test message and test values for the parsers.

```
<patterndb version='4' pub_date='2010-10-17'>
 <ruleset name='ssh' id='123456678'>
 <pattern>ssh</pattern>
 <rules>
 <rule provider='me' id='182437592347598' class='system'>
 <patterns>
 <pattern>Accepted @QSTRING:SSH.AUTH_METHOD: @
for@QSTRING:SSH_USERNAME: @from\ @QSTRING:SSH_CLIENT_ADDRESS: @port @NUMBER:SSH_PORT_NUMBER:@
ssh2</pattern>
 </patterns>
 <examples>
 <example>
 <test_message>Accepted password for sampleuser from 10.50.0.247
port 42156 ssh2</test_message>
 <test_values>
 <test_value name="SSH.AUTH_METHOD">password</test_value>
 <test_value name="SSH_USERNAME">sampleuser</test_value>
 <test_value
name="SSH_CLIENT_ADDRESS">10.50.0.247</test_value>
 <test_value name="SSH_PORT_NUMBER">42156</test_value>
 </test_values>
 </example>
 </examples>
 </rule>
 </rules>
 </ruleset>
</patterndb>
```



## Chapter 14. Statistics of syslog-ng

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

The following is a sample log statistics message for a configuration that has a single source (*s\_local*) and a network and a local file destination (*d\_network* and *d\_local*, respectively). All incoming messages are sent to both destinations.

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

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

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

The statistics include a list of source groups and destinations, as well as the number of processed messages for each. The verbosity of the statistics can be set using the `stats_level()` option. For details, see *Section 9.2, Global options (p. 137)*. An example output is shown below.

```
src.internal;s_all#0;;a;processed;6445
src.internal;s_all#0;;a;stamp;1268989330
destination;df_auth;;a;processed;404
destination;df_news_dot_notice;;a;processed;0
destination;df_news_dot_err;;a;processed;0
destination;d_ssbb;;a;processed;7128
destination;df_uucp;;a;processed;0
source;s_all;;a;processed;7128
destination;df_mail;;a;processed;0
destination;df_user;;a;processed;1
destination;df_daemon;;a;processed;1
destination;df_debug;;a;processed;15
destination;df_messages;;a;processed;54
destination;dp_xconsole;;a;processed;671
dst.tcp;d_network#0;10.50.0.111:514;a;dropped;5080
dst.tcp;d_network#0;10.50.0.111:514;a;processed;7128
```



```
dst.tcp;d_network#0;10.50.0.111:514;a;stored;2048
destination;df_syslog;;a;processed;6724
destination;df_facility_dot_warn;;a;processed;0
destination;df_news_dot_crit;;a;processed;0
destination;df_lpr;;a;processed;0
destination;du_all;;a;processed;0
destination;df_facility_dot_info;;a;processed;0
center;;received;a;processed;0
destination;df_kern;;a;processed;70
center;;queued;a;processed;0
destination;df_facility_dot_err;;a;processed;0
```

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

1. The type of the object (for example `dst.file`, `tag`, `src.facility`)
2. The ID of the object used in the syslog-ng configuration file, for example `d_internal` or `source.src_tcp`. The `#0` part means that this is the first destination in the destination group.
3. The instance ID (destination) of the object, for example the filename of a file destination, or the name of the application for a program source or destination.
4. The status of the object. One of the following:
  - `a` - active. At the time of 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.
  - `o` - This object was once active, but stopped receiving messages. (For example a dynamic object may disappear and become orphan.)
5. The type of the statistics:
  - `processed`: The number of messages that successfully reached their destination.
  - `dropped`: The number of dropped messages — syslog-ng OSE could not send the messages to the destination and the output buffer got full, so messages were lost.
  - `stored`: The number of messages stored in the message queue, waiting to be sent to the destination.
  - `suppressed`: The number of suppressed messages (if the `suppress()` feature is enabled).
  - `stamp`: The UNIX timestamp of the last message sent to the destination.
6. The number of such messages.



**Note**

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

When receiving messages with non-standard facility values (that is, higher than 23), these messages will be listed as `other` facility instead of their facility number.



# Chapter 15. Troubleshooting syslog-ng

This chapter provides tips and guidelines about troubleshooting problems related to syslog-ng.



## Tip

As a general rule, first try to get logging the messages to a local file. Once this is working, you know that syslog-ng is running correctly and receiving messages, and you can proceed to forwarding the messages to the server.

If the syslog-ng server does not receive the messages, use `tcpdump` or a similar packet sniffer tool on the client to verify that the messages are sent correctly, and on the server to verify that it receives the messages.

If syslog-ng is closing the connections for no apparent reason, be sure to check the log messages of syslog-ng. You might also want to run syslog-ng with the `--verbose` or `--debug` command-line options for more-detailed log messages. Starting from syslog-ng OSE version 3.1, you can enable these messages without restarting syslog-ng using the `syslog-ctl verbose --set=on` command. For details, see the `syslog-ctl` man page at [syslog-ctl\(1\)](#) (p. 216).

Similarly, build up encrypted connections step-by-step: first create a working unencrypted (for example TCP) connection, then add TLS encryption, and finally client authentication if needed.

## 15.1. 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 syslog-ng does its best to avoid message loss. Usually losing messages can be avoided with careful planning and proper configuration of syslog-ng and the hosts running syslog-ng. The following list shows the possible locations where messages may be lost, and provides methods to minimize the risk of losing messages.



## Note

The following list covers the main possibilities of losing messages, but does not take into account the possible use of flow-control (for details, see [Section 8.2, "Managing incoming and outgoing messages with flow-control" \(p. 125\)](#)). This topic will be addressed in more detail in the future releases of this guide.

- *Between the application and the syslog-ng client:* Make sure to use an appropriate source to receive the logs from the application (for example from `/dev/log`). For example, use `unix-stream` instead of `unix-dgram` whenever possible.
- *When syslog-ng is sending messages:* If syslog-ng cannot send messages to the destination and the output buffer gets full, syslog-ng will drop messages. The number of dropped messages is displayed per destination in the log message statistics of syslog-ng (for details, see [Chapter 14, "Statistics of syslog-ng" \(p. 191\)](#)).
- *On the network:* When transferring messages using the UDP protocol, messages may be lost without any notice or feedback — such is the nature of the UDP protocol. Always use the TCP protocol to transfer messages over the network whenever possible.
- *In the socket receive buffer:* When transferring messages using the UDP protocol, the UDP datagram (that is, the message) that reaches the receiving host placed in a memory area called the `socket receive buffer`. If the host receives more messages than it can process, this area overflows, and the kernel drops messages without letting syslog-ng know about it. Using TCP instead of UDP prevents this issue. If you must use the UDP protocol, increase the size of the receive buffer using the `so_rcvbuf()` option.



- *When syslog-ng is receiving messages:* The receiving syslog-ng (for example the syslog-ng server or relay) may drop messages if the fifo of the destination file gets full. The number of dropped messages is displayed per destination in the log message statistics of syslog-ng (for details, see *Chapter 14, Statistics of syslog-ng (p. 191)*).
- *When the destination cannot handle large load:* When syslog-ng is sending messages at a high rate into an SQL database, a file, or another destination, it is possible that the destination cannot handle the load, and processes the messages slowly. As a result, the buffers of syslog-ng fill up, syslog-ng cannot process the incoming messages, and starts to loose messages. For details, see the previous entry. Use the *throttle* parameter to avoid this problem.
- *As a result of an unclean shutdown of the syslog-ng server:* If the host running the syslog-ng server experiences an unclean shutdown, it takes time until the clients realize that the connection to the syslog-ng server is down. Messages that are put into the output TCP buffer of the clients during this period are not sent to the server.

## 15.2. Procedure – Creating syslog-ng core files

### Purpose:

When syslog-ng crashes for some reason, it can create a core file that contains important troubleshooting information. To enable core files, complete the following procedure:

### Steps:

Step 1. Core files are produced only if the *maximum core file size* ulimit is set to a high value in the init script of syslog-ng. Add the following line to the init script of syslog-ng:

```
ulimit -c unlimited
```

Step 2. Verify that syslog-ng has permissions to write the directory it is started from, for example /opt/syslog-ng/sbin/.

Step 3. If syslog-ng crashes, it will create a core file in the directory syslog-ng was started from.

Step 4. To test that syslog-ng can create a core file, you can create a crash manually. For this, determine the PID of syslog-ng (for example using the ps -All|grep syslog-ng command), then issue the following command: kill -ABRT <syslog-ng pid>

This should create a core file in the current working directory.

## 15.3. Collecting debugging information with strace, truss, or tusc

To properly troubleshoot certain situations, it can be useful to trace which system calls syslog-ng OSE performs. How this is performed depends on the platform running syslog-ng OSE. In general, note the following points:

- When syslog-ng OSE is started, a supervisor process might stay in the foreground, while the actual syslog-ng daemon goes to the background. Always trace the background process.
- Apart from the system calls, the time between two system calls can be important as well. Make sure that your tracing tool records the time information as well. For details on how to do that, refer to the manual page of your specific tool (for example, strace on Linux, or truss on Solaris and BSD).



- Run your tracing tool in verbose mode, and if possible, set it to print long output strings, so the messages are not truncated.
- When using `strace`, also record the output of `lsof` to see which files are accessed.

The following are examples for tracing system calls of syslog-ng on some platforms. The output is saved into the `/tmp/syslog-ng-trace.txt` file, sufficed with the PID of the related syslog-ng process. The path of the syslog-ng binary assumes that you have installed syslog-ng OSE from the official syslog-ng OSE binaries available at the Balabit website — native distribution-specific packages may use different paths.

- *Linux:* `strace -o /tmp/trace.txt -s256 -ff -ttT /opt/syslog-ng/sbin/syslog-ng -f /opt/syslog-ng/etc/syslog-ng.conf -Fdv`
- *HP-UX:* `tusc -f -o /tmp/syslog-ng-trace.txt -T /opt/syslog-ng/sbin/syslog-ng`
- *IBM AIX and Solaris:* `truss -f -o /tmp/syslog-ng-trace.txt -r all -w all -u libc:: /opt/syslog-ng/sbin/syslog-ng -d -d -d`

**Tip**

To execute these commands on an already running syslog-ng OSE process, use the `-p <pid_of_syslog-ng>` parameter.

## 15.4. Running a failure script

When syslog-ng is abnormally terminated, it can execute a user-created failure script. This can be used for example to send an automatic e-mail notification. The script must be located at `/opt/syslog-ng/sbin/syslog-ng-failure`.

## 15.5. Stopping syslog-ng

To avoid problems, always use the init scripts to stop syslog-ng (`/etc/init.d/syslog-ng stop`), instead of using the `kill` command. This is especially true on Solaris and HP-UX systems, here use `/etc/init.d/syslog stop`.



# Chapter 16. Best practices and examples

This chapter discusses some special examples and recommendations.

## 16.1. General recommendations

This section provides general tips and recommendations on using syslog-ng. Some of the recommendations are detailed in the subsequent sections.

- Do not base the separation of log messages into different files on the *facility* parameter. As several applications and processes can use the same facility, the facility does not identify the application that sent the message. By default, the *facility* parameter is not even included in the log message itself. In general, sorting the log messages into several different files can make finding specific log messages difficult. If you must create separate log files, use the application name.
- Standard log messages include the local time of the sending host, without any time zone information. It is recommended to replace this timestamp with an ISODATE timestamp, because the ISODATE format includes the year and timezone as well. To convert all timestamps to the ISODATE format, include the following line in the syslog-ng configuration file:

```
options {ts_format(iso) ;};
```

- Resolving the IP addresses of the clients to domain names can decrease the performance of syslog-ng. For details, see *Section 16.4, Using name resolution in syslog-ng* (p. 197).

## 16.2. Handling lots of parallel connections

When syslog-ng is receiving messages from a large number of TCP or unix-stream connections, the CPU usage of syslog-ng might increase even if the number of messages is low. By default, syslog-ng processes every message when it is received. To reduce the CPU usage, process the incoming messages in batches. To accomplish this, instruct syslog-ng to wait for a short time before processing a message. During this period additional messages might arrive that can be processed together with the original message. To process log messages in batches, set the *time\_sleep()* option (measured in milliseconds) to a non-zero value. Include the following line in your syslog-ng configuration:

```
options { time_sleep(20) ;};
```



### Note

It is not recommended to increase the *time\_sleep()* parameter above 100ms, as that might distort timestamps, slow down syslog-ng, and cause messages to be dropped.

When modifying the *time\_sleep()* option, also adjust the *log\_fetch\_limit()* and *log\_fifo\_size()* options accordingly.

The *max\_connections()* parameter limits the number of parallel connections for the source.



If adjusting the `time_sleep()` option is not desired for some reason, an alternative solution is to use `unix-stream()`, `udp()` and `unix-dgram()` sources instead of `tcp()` connections.

### 16.3. Handling large message load

This section provides tips on optimizing the performance of syslog-ng. Optimizing the performance is important for syslog-ng hosts that handle large traffic.

- Disable DNS resolution, or resolve hostnames locally. For details, see *Section 16.4, Using name resolution in syslog-ng* (p. 197).
- Enable flow-control for the TCP sources. For details, see *Section 8.2, Managing incoming and outgoing messages with flow-control* (p. 125).
- Do not use the `user tty()` 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 *Section 8.3.4, Using wildcards, special characters, and regular expressions in filters* (p. 131).
- When receiving messages using the UDP protocol, increase the size of the UDP receive buffer on the receiver host (that is, the syslog-ng OSE server or relay receiving the messages). Note that on certain platforms, for example, on Red Hat Enterprise Linux 5, even low message load (~200 messages per second) can result in message loss, unless the `so_recvbuf()` option of the source is increased. In such cases, you will need to increase the `net.core.rmem_max` parameter of the host (for example, to `1024000`), but do not modify `net.core.rmem_default` parameter.

As a general rule, increase the `so_recvbuf()` 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_recvbuf()` at least to `2 097 152` bytes. For information about sizing and modifying the UDP buffer, see <http://www.29west.com/docs/THPM/udp-buffer-sizing.html>.

### 16.4. Using name resolution in syslog-ng

The syslog-ng application can resolve the hostnames of the clients and include them in the log messages. However, the performance of syslog-ng is severely degraded if the domain name server is unaccessible or slow. Therefore, it is not recommended to resolve hostnames in syslog-ng. If you must use name resolution from syslog-ng, consider the following:

- Use DNS caching. Verify that the DNS cache is large enough to store all important hostnames. (By default, the syslog-ng DNS cache stores `1007` entries.)

```
options { dns_cache(2000); };
```

- If the IP addresses of the clients change only rarely, set the expiry of the DNS cache large.

```
options { dns_cache_expire(87600); };
```

- If possible, resolve the hostnames locally. For details, see *Procedure 16.4.1, Resolving hostnames locally* (p. 198).

**Note**

Domain name resolution is important mainly in relay and server mode.

### 16.4.1. Procedure – Resolving hostnames locally

**Purpose:**

Resolving hostnames locally enables you to display hostnames in the log files for frequently used hosts, without having to rely on a DNS server. The known IP address – hostname pairs are stored locally in a file. In the log messages, syslog-ng will replace the IP addresses of known hosts with their hostnames. To configure local name resolution, complete the following steps:

**Steps:**

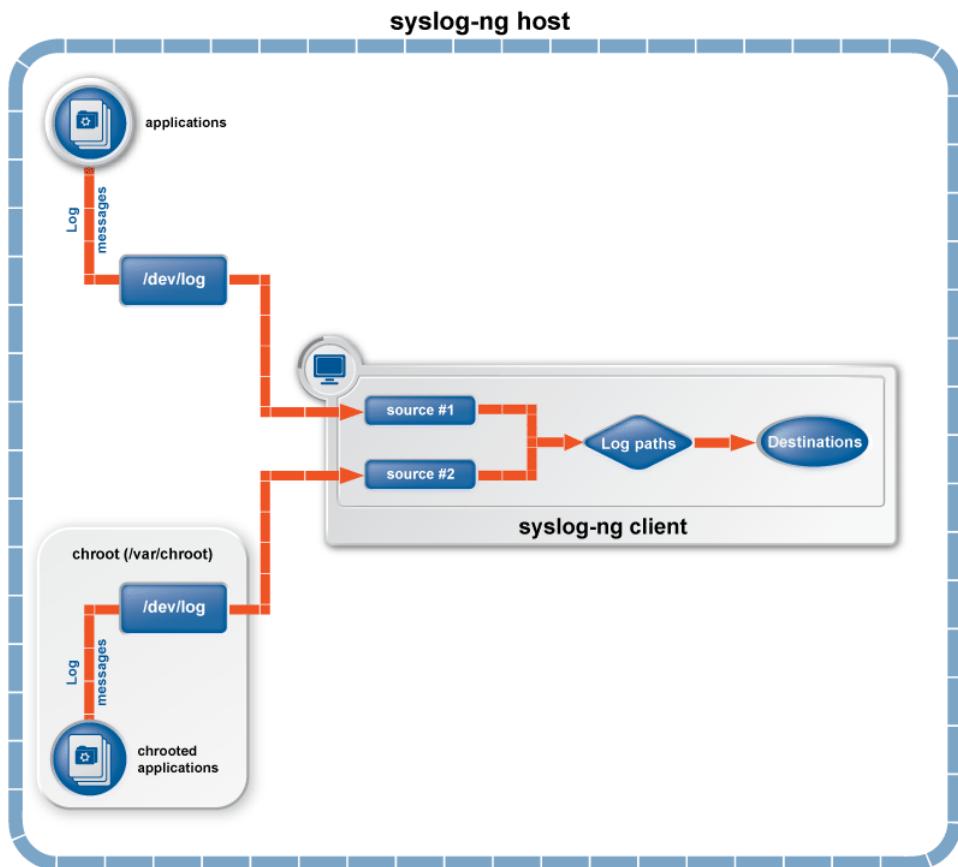
- Step 1. Add the hostnames and the respective IP addresses to the file used for local name resolution. On Linux and UNIX systems, this is the `/etc/hosts` file. Consult the documentation of your operating system for details.
- Step 2. Instruct syslog-ng to resolve hostnames locally. Set the `use_dns()` option of syslog-ng to `persist_only`.
- Step 3. Set the `dns_cache_hosts()` option to point to the file storing the hostnames.

```
options {
 use_dns(persist_only);
 dns_cache_hosts(/etc/hosts); };
```

### 16.5. Procedure – Collecting logs from chroot

**Purpose:**

To collect logs from a chroot using a syslog-ng client running on the host, complete the following steps:



*Figure 16.1. Collecting logs from chroot*

### Steps:

- Step 1. Create a `/dev` directory within the chroot. The applications running in the chroot send their log messages here.
- Step 2. Create a local source in the configuration file of the syslog-*ng* application running outside the chroot. This source should point to the `/dev/log` file within the chroot (for example to the `/chroot/dev/log` directory).
- Step 3. Include the source in a log statement.



#### Note

You need to set up timezone information within your chroot as well. This usually means creating a symlink to `/etc/localtime`.



## Appendix 1. The syslog-*ng* manual pages



## Name

syslog-*ng* — syslog-*ng* system logger application

## Synopsis

`syslog-ng [options]`

## Description

NOTE: This manual page covers both editions of syslog-*ng*: syslog-*ng* Open Source Edition and the commercial syslog-*ng* Premium Edition. Features that are only included in the Premium Edition are marked with an asterisk (\*). For details, see [\*the official syslog-\*ng\* website\*](#).

This manual page is only an abstract; for the complete documentation of syslog-*ng*, see [\*The syslog-\*ng\* Administrator Guide\*](#).

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

## Options

<code>--cfgfile &lt;file&gt; or -f &lt;file&gt;</code>	Use the specified configuration file.
<code>--chroot &lt;dir&gt; or -C &lt;dir&gt;</code>	Change root to the specified directory after reading the configuration file. The directory must be set up accordingly. Note that it is not possible to reload the syslog- <i>ng</i> configuration after chrooting.
<code>--debug or -d</code>	Start syslog- <i>ng</i> in debug mode.
<code>--enable-core</code>	Enable syslog- <i>ng</i> to write core files in case of a crash to help support and debugging.
<code>--fd-limit</code>	Set the minimal number of required file descriptors (fd-s); this sets how many files syslog- <i>ng</i> can keep open simultaneously. Default value: 4096. Note that this does not override the global ulimit setting of the host.
<code>--foreground or -F</code>	Do not daemonize, run in the foreground.
<code>--group &lt;group&gt; or -g &lt;group&gt;</code>	Switch to the specified group after initializing the configuration file.
<code>--help or -h</code>	Display a brief help message.
<code>--no-caps</code>	Run syslog- <i>ng</i> as root, without capability-support. This is the default behavior. On Linux, it is possible to run syslog- <i>ng</i> as non-root with capability-support if syslog- <i>ng</i> was compiled with the <code>--enable-linux-caps</code> option enabled. (Execute <code>syslog-<i>ng</i> --version</code> to display the list of enabled build parameters.)



--persist-file <persist-file> or -R <persist-file>	Set the path and name of the <code>syslog-ng.persist</code> file where the persistent options and data are stored.
--pidfile <pidfile> or -p <pidfile>	Set path to the PID file where the pid of the main process is stored.
--process-mode <pidfile>	Sets how to run <code>syslog-ng</code> : in the <i>foreground</i> (mainly used for debugging), in the <i>background</i> as a daemon, or in <i>safe-background</i> mode. By default, <code>syslog-ng</code> runs in <i>safe-background</i> mode. This mode creates a supervisor process called <i>supervising syslog-ng</i> , that restarts <code>syslog-ng</code> if it crashes.
--stderr or -e	Log internal messages of <code>syslog-ng</code> to stderr. Mainly used for debugging purposes in conjunction with the <code>--foreground</code> option. If not specified, <code>syslog-ng</code> will log such messages to its internal source.
--syntax-only or -s	Verify that the configuration file is syntactically correct and exit.
--user <user> or -u <user>	Switch to the specified user after initializing the configuration file (and optionally chrooting). Note that it is not possible to reload the <code>syslog-ng</code> configuration if the specified user has no privilege to create the <code>/dev/log</code> file.
--verbose or -v	Enable verbose logging used to troubleshoot <code>syslog-ng</code> .
--version or -V	Display version number and compilation information.

## Files

`/opt/syslog-ng/etc/syslog-ng/`

`/opt/syslog-ng/etc/syslog-ng/syslog-ng.conf`

## See also

[`syslog-ng.conf\(5\)`](#)

[The `syslog-ng` Administrator Guide](#)

If you experience any problems or need help with `syslog-ng`, visit the [`syslog-ng mailing list`](#)

For news and notifications about the documentation of `syslog-ng`, visit the [BalaBit Documentation Blog](#).

## Author

This manual page was written by the BalaBit Documentation Team <[documentation@balabit.com](mailto:documentation@balabit.com)>.



## Copyright

Copyright © 2000-2011 BalaBit IT Security Ltd. Published under the Creative Commons Attribution-Noncommercial-No Derivative Works (by-nc-nd) 3.0 license. For details, see <http://creativecommons.org/>. The latest version is always available at <http://www.balabit.com/support/documentation>.



## Name

syslog-*ng*.conf — syslog-*ng* configuration file

## Synopsis

syslog-*ng*.conf

## Description

NOTE: This manual page covers both editions of syslog-*ng*: syslog-*ng* Open Source Edition and the commercial syslog-*ng* Premium Edition. Features that are only included in the Premium Edition are marked with an asterisk (\*). For details, see [the official syslog-\*ng\* website](#).

This manual page is only an abstract; for the complete documentation of syslog-*ng*, see [The syslog-\*ng\* Administrator Guide](#).

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

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

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

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

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

## Configuring syslog-*ng*

- The main body of the configuration file consists of object definitions: sources, destinations, logpaths define which log message are received and where they are sent. All identifiers, option names and attributes, and any other strings used in the syslog-*ng* configuration file are case sensitive. Objects must be defined before they are referenced in another statement. Object definitions (also called statements) have the following syntax:

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

- *Type of the object*: One of *source*, *destination*, *log*, *filter*, *parser*, *rewrite rule*, or *template*.



- *Identifier of the object*: A unique name identifying the object. When using a reserved word as an identifier, enclose the identifier in quotation marks.



**Tip**

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

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

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

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

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

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

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

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

For example, the following source statement has three options; the first two options (`file()` and `follow_freq()`) have a single parameter, while the third one (`flags()`) has two parameters:

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

Objects may have required and optional parameters. Required parameters are positional, meaning that they must be specified in a defined order. Optional parameters can be specified in any order using the `option(value)` format. If a parameter (optional or required) is not specified, its default value is used. The parameters and their default values are listed in the reference section of the particular object.



**Example 1.1. Using required and optional parameters**

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

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

- Some options are global options, or can be set globally, for example, whether syslog-ng OSE should use DNS resolution to resolve IP addresses. Global options are detailed in *Chapter 9, Global options of syslog-ng OSE* (p. 137).



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

- All identifiers, attributes, and any other strings used in the syslog-ng configuration file are case sensitive.
- Objects can be used before definition.
- To add comments to the configuration file, start a line with # and write your comments. These lines are ignored by syslog-ng.

```
Comment: This is a stream source
source s_demo_stream {
 unix-stream("/dev/log" 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 { tcp(ip(127.0.0.1) port(1999)); };
destination d_tcp { tcp("10.1.2.3" port(1999); localport(999)); };
log { source(s_localhost); destination(d_tcp); };
```

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

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

The sources, destinations, and filters available in syslog-ng are listed below. For details, see [The syslog-ng Administrator Guide](#).

Name	Description
<i>internal()</i>	Messages generated internally in syslog-ng.
<i>file()</i>	Opens the specified file and reads messages.
<i>pipe(), fifo</i>	Opens the specified named pipe and reads messages.
<i>pacct()</i>	Reads messages from the process accounting logs on Linux.
<i>program()</i>	Opens the specified application and reads messages from its standard output.
<i>sun-stream(), sun-streams()</i>	Opens the specified STREAMS device on Solaris systems and reads incoming messages.
<i>syslog()</i>	Listens for incoming messages using the new <i>IETF-standard syslog protocol</i> .



Name	Description
<code>system()</code>	Automatically detects which platform syslog-ng OSE is running on, and collects the native log messages of that platform.
<code>tcp(), tcp6()</code>	Listens on the specified TCP port for incoming messages using the <i>BSD-syslog protocol</i> over IPv4 and IPv6 networks, respectively.
<code>udp(), udp6()</code>	Listens on the specified UDP port for incoming messages using the <i>BSD-syslog protocol</i> over IPv4 and IPv6 networks, respectively.
<code>unix-dgram()</code>	Opens the specified unix socket in <i>SOCK_DGRAM</i> mode and listens for incoming messages.
<code>unix-stream()</code>	Opens the specified unix socket in <i>SOCK_STREAM</i> mode and listens for incoming messages.

Table 1.1. Source drivers available in syslog-ng

Name	Description
<code>file()</code>	Writes messages to the specified file.
<code>fifo(), pipe()</code>	Writes messages to the specified named pipe.
<code>program()</code>	Forks and launches the specified program, and sends messages to its standard input.
<code>sql()</code>	Sends messages into an SQL database. In addition to the standard syslog-ng packages, the <code>sql()</code> destination requires database-specific packages to be installed. Refer to the section appropriate for your platform in <i>Chapter 3, Installing syslog-ng (p. 16)</i> .
<code>syslog()</code>	Sends messages to the specified remote host using the <i>IETF-syslog protocol</i> . The IETF standard supports message transport using the UDP, TCP, and TLS networking protocols.
<code>tcp() and tcp6()</code>	Sends messages to the specified TCP port of a remote host using the <i>BSD-syslog protocol</i> over IPv4 and IPv6, respectively.
<code>udp() and udp6()</code>	Sends messages to the specified UDP port of a remote host using the <i>BSD-syslog protocol</i> over IPv4 and IPv6, respectively.
<code>unix-dgram()</code>	Sends messages to the specified unix socket in <i>SOCK_DGRAM</i> style (BSD).
<code>unix-stream()</code>	Sends messages to the specified unix socket in <i>SOCK_STREAM</i> style (Linux).
<code>useratty()</code>	Sends messages to the terminal of the specified user, if the user is logged in.

Table 1.2. Destination drivers available in syslog-ng

Name	Description
<code>facility()</code>	Filter messages based on the sending facility.
<code>filter()</code>	Call another filter function.
<code>host()</code>	Filter messages based on the sending host.
<code>level() or priority()</code>	Filter messages based on their priority.
<code>match()</code>	Use a regular expression to filter messages based on a specified header or content field.
<code>message()</code>	Use a regular expression to filter messages based on their content.
<code>netmask()</code>	Filter messages based on the IP address of the sending host.
<code>program()</code>	Filter messages based on the sending application.



Name	Description
<code>source()</code>	Select messages of the specified syslog-ng OSE source statement.
<code>tags()</code>	Select messages having the specified tag.

Table 1.3. Filter functions available in *syslog-ng* OSE

## Files

/opt/syslog-ng/etc/syslog-ng/

/opt/syslog-ng/etc/syslog-ng/syslog-ng.conf

## See also

[\*syslog-ng\(8\)\*](#)

[\*The syslog-ng Administrator Guide\*](#)

If you experience any problems or need help with syslog-ng, visit the [\*syslog-ng mailing list\*](#)

For news and notifications about the documentation of syslog-ng, visit the [\*BalaBit Documentation Blog\*](#).

## Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

## Copyright

Copyright © 2000-2011 BalaBit IT Security Ltd. Published under the Creative Commons Attribution-Noncommercial-No Derivative Works (by-nc-nd) 3.0 license. For details, see <http://creativecommons.org/>. The latest version is always available at <http://www.balabit.com/support/documentation>.



## Name

`pdbtool` — An application to test and convert syslog-*ng* pattern database rules

## Synopsis

`pdbtool [command] [options]`

## Description

This manual page is only an abstract; for the complete documentation of syslog-*ng* and `pdbtool`, see [\*The syslog-\*ng\* Administrator Guide\*](#).

The syslog-*ng* application can match the contents of the log messages to a database of predefined message patterns (also called patterndb). By comparing the messages to the known patterns, syslog-*ng* is able to identify the exact type of the messages, tag the messages, and sort them into message classes. The message classes can be used to classify the type of the event described in the log message. The functionality of the pattern database is similar to that of the logcheck project, but the syslog-*ng* approach is faster, scales better, and is much easier to maintain compared to the regular expressions of logcheck.

The `pdbtool` application is a utility that can be used to:

- test message patterns;
- convert an older pattern database to the latest database format;
- merge pattern databases into a single file;
- dump the RADIX tree built from the pattern database (or a part of it) to explore how the pattern matching works.

## The dump command

`dump [options]`

Display the RADIX tree built from the patterns. This shows how are the patterns represented in syslog-*ng* and it might also help to track down pattern-matching problems. The dump utility can dump the tree used for matching the PROGRAM or the MSG parts.

- |                                   |                                                                                                |
|-----------------------------------|------------------------------------------------------------------------------------------------|
| <code>--pdb or -p</code>          | Name of the pattern database file to use.                                                      |
| <code>--program or -P</code>      | Displays the RADIX tree built from the patterns belonging to the <i>\$PROGRAM</i> application. |
| <code>--program-tree or -T</code> | Display the <i>\$PROGRAM</i> tree.                                                             |

Example and sample output:

```
pdbtool dump -p patterndb.xml -P 'sshd'
```

```
'p'
 'assword for'
 @QSTRING:@
 'from'
```



```
@QSTRING:@
 'port'
 @NUMBER:@ rule_id='fc49054e-75fd-11dd-9bba-001e6806451b'
 ' ssh' rule_id='fc55cf86-75fd-11dd-9bba-001e6806451b'
 '2' rule_id='fc4b7982-75fd-11dd-9bba-001e6806451b'
'ublickey for'
@QSTRING:@
 'from'
@QSTRING:@
 'port'
 @NUMBER:@ rule_id='fc4d377c-75fd-11dd-9bba-001e6806451b'
 ' ssh' rule_id='fc5441ac-75fd-11dd-9bba-001e6806451b'
 '2' rule_id='fc44a9fe-75fd-11dd-9bba-001e6806451b'
```

## 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-csv or -C	Print the debugging information returned by the <code>--debug-pattern</code> option as comma-separated values.
--debug-pattern or -D	Print debugging information about the pattern matching. See also the <code>--debug-csv</code> 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 ( <code>stdin</code> ), specify a hyphen (-) character instead of a filename.
--filter=<filter-expression> or -F	Print only messages matching the specified <code>syslog-ng</code> filter expression.
--message or -M	The text of the log message to match (only the <code>\$MESSAGE</code> part without the <code>syslog</code> headers).
--pdb or -p	Name of the pattern database file to use.



--program or -P	Name of the program to use, as contained in the \$PROGRAM part of the syslog message.
--template=<template-expression> or -T	A syslog- <i>ng</i> template expression that is used to format the output messages.

Example: The following command checks if the patterndb.xml file recognizes the *Accepted publickey for myuser from 127.0.0.1 port 59357 ssh2* message:

```
pdbtool match -p patterndb.xml -P sshd -M "Accepted publickey for myuser from 127.0.0.1 port 59357 ssh2"
```

The following example applies the sshd.pdb pattern database file to the log messages stored in the /var/log/messages file, and displays only the messages that received a *useracct* tag.

```
pdbtool match -p sshd.pdb \
 -file /var/log/messages \
 -filter 'tags("usracct");'
```

## The merge command

merge [options]

Use the *merge* command to combine separate pattern database files into a single file (pattern databases are usually stored in separate files per applications to simplify maintenance). If a file uses an older database format, it is automatically updated to the latest format (V3). See the *The syslog-*ng* Administrator Guide* for details on the different pattern database versions.

--directory or -D	The directory that contains the pattern database XML files to be merged.
--glob or -G	Specify filenames to be merged using a glob pattern, for example, using wildcards. For details on glob patterns, see man <i>glob</i> . 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.

Example:

```
pdbtool merge --recursive --directory /home/me/mypatterns/ --pdb
/var/lib/syslog-ng/patterndb.xml
```

Currently it is not possible to convert a file without merging, so if you only want to convert an older pattern database file to the latest format, you have to copy it into an empty directory.

## The patternize command

patternize [options]

Automatically create a pattern database from a log file containing a large number of log messages. The resulting pattern database is printed to the standard output (stdout). The *pdbtool patternize* command uses a data clustering technique to find similar log messages and replacing the differing parts with @ESTRING:: @ parsers.



For details on pattern databases and message parsers, see the [\*The syslog-ng Administrator Guide\*](#). The `patternize` command is available only in syslog-ng OSE version 3.2 and later.

<code>--file=&lt;path&gt; or -f</code>	The logfile containing the log messages to create patterns from. To receive the log messages from the standard input (stdin), use <code>-</code> .
<code>--iterate-outliers or -o</code>	Recursively iterate on the log lines to cover as many log messages with patterns as possible.
<code>--named-parsers or -n</code>	The number of example log messages to include in the pattern database for every pattern. Default value: <code>1</code>
<code>--samples=&lt;number-of-samples&gt;</code>	Include a generated name in the parsers, for example, <code>.dict.string1</code> , <code>.dict.string2</code> , and so on.
<code>--support=&lt;number&gt; or -S</code>	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 <code>--support=3.0</code> 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 <code>patternize</code> does not create enough patterns, try to decrease the support value. Default value: <code>4.0</code>

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 `xmlint` application installed. The `test` command is available only in syslog-ng OSE version 3.2 and later.

`--validate` Validate a pattern database XML file.

Example:

```
pdbtool test --validate /home/me/mypatterndb.pdb
```

## Files

`/opt/syslog-ng/bin/pdbtool`

`/opt/syslog-ng/etc/syslog-ng/syslog-ng.conf`

## See also

[\*The syslog-ng Administrator Guide\*](#)

[\*syslog-ng.conf\(5\)\*](#)



## [syslog-\*ng\*\(8\)](#)

If you experience any problems or need help with syslog-*ng*, visit the [syslog-\*ng\* mailing list](#)

For news and notifications about the documentation of syslog-*ng*, visit the [BalaBit Documentation Blog](#).

### **Author**

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

### **Copyright**

Copyright © 2000-2011 BalaBit IT Security Ltd. Published under the Creative Commons Attribution-Noncommercial-No Derivative Works (by-nc-nd) 3.0 license. For details, see <http://creativecommons.org/>. The latest version is always available at <http://www.balabit.com/support/documentation>.



## Name

loggen — Generate syslog messages at a specified rate

## Synopsis

```
loggen [options]target [port]
```

## Description

NOTE: The loggen application is distributed with the syslog-*ng* system logging application, and is usually part of the syslog-*ng* package. The latest version of the syslog-*ng* application is available at [the official syslog-\*ng\* website](#).

This manual page is only an abstract; for the complete documentation of syslog-*ng*, see [The syslog-\*ng\* Administrator Guide](#).

The loggen application is tool to test and stress-test your syslog server and the connection to the server. It can send syslog messages to the server at a specified rate, using a number of connection types and protocols.

## Options

--csv or -C	Send statistics of the sent messages to stdout as CSV. This can be used for plotting the message rate.
--dgram or -D	Use datagram socket (UDP or unix-dgram) to send the messages to the target.
--help or -h	Display a brief help message.
--inet or -i	Use the TCP (by default) or UDP (when used together with the -dgram option) protocol to send the messages to the target.
--interval <seconds> or -I <seconds>	The number of seconds loggen will run. Default value: 10
--no-framing or -F	Do not use the framing of the IETF-syslog protocol style, even if the <i>syslog-proto</i> option is set.
--rate <message/second> or -r <message/second>	The number of messages generated per second. Default value: 1000
--read-file or -R	Read the messages from a file and send them to the target. See also the <i>--skip-tokens</i> option.
--size or -s	The size of a syslog message in bytes. Default value: 256
--skip-tokens	Skip the specified number of space-separated tokens (words) at the beginning of every line. For example, if the messages in the file look like <i>foo bar message</i> , <i>--skip-tokens 2</i> skips the <i>foo bar</i> part of the line, and sends only the <i>message</i> part. Works only when used together with the <i>--read-file</i> parameter.
--stream or -S	Use a stream socket (TCP or unix-stream) to send the messages to the target.



--syslog-proto or -P	Use the new IETF-syslog message format as specified in RFC5424. By default, loggen uses the legacy BSD-syslog message format (as described in RFC3164). See also the <code>--no-framing</code> option.
--unix or -x	Use a UNIX domain socket to send the messages to the target.
--use-ssl or -U	Use an SSL-encrypted channel to send the messages to the target. Note that it is not possible to check the certificate of the target, or to perform mutual authentication.
--verbose or -V	Display the actual speed of sending messages in messages/second.

## Example

The following command generates 100 messages per second for ten minutes, and sends them to port 2010 of the localhost via TCP. Each message is 300 bytes long.

```
loggen --size 300 --rate 100 --interval 600 127.0.0.1 2010
```

The following command is similar to the one above, but uses the UDP protocol.

```
loggen --inet --dgram --size 300 --rate 100 --interval 600 127.0.0.1 2010
```

## Files

/opt/syslog-ng/bin/loggen

## See also

[syslog-\*ng\*.conf\(5\)](#)

[The syslog-\*ng\* Administrator Guide](#)

If you experience any problems or need help with loggen or syslog-*ng*, visit the [syslog-\*ng\* mailing list](#)

## Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

## Copyright

Copyright © 2000-2011 BalaBit IT Security Ltd. Published under the Creative Commons Attribution-Noncommercial-No Derivative Works (by-nc-nd) 3.0 license. For details, see <http://creativecommons.org/>. The latest version is always available at <http://www.balabit.com/support/documentation>.



## Name

syslog-ctl — Display message statistics and enable verbose, debug and trace modes in syslog-Open Source Edition

## Synopsis

`syslog-ctl [command] [options]`

## Description

NOTE: The syslog-ctl application is distributed with the syslog-Open Source Edition system logging application, and is usually part of the syslog package. The latest version of the syslog application is available at [the official syslog website](#).

This manual page is only an abstract; for the complete documentation of syslog-Open Source Edition, see [The syslog-Open Source Edition Administrator Guide](#).

The `syslog-ctl` application is a utility that can be used to:

- enable/disable various syslog messages for troubleshooting;
- display statistics about the processed messages.

## Enabling troubleshooting messages

`command [options]`

Use the `syslog-ctl <command> --set=on` command to display verbose, trace, or debug messages. If you are trying to solve configuration problems, the debug (and occasionally trace) messages are usually sufficient; debug messages are needed mostly for finding software errors. After solving the problem, do not forget to turn these messages off using the `syslog-ctl <command> --set=off`. Note that enabling debug messages does not enable verbose and trace messages.

Use `syslog-ctl <command>` without any parameters to display whether the particular type of messages are enabled or not.

If you need to use a non-standard control socket to access syslog-Open Source Edition, use the `syslog-ctl <command> --set=on --control=<socket>` command to specify the socket to use.

- |         |                                                                                                                                                                                                                                                                                                        |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| verbose | Print verbose messages. If syslog-Open Source Edition was started with the <code>--stderr</code> or <code>-e</code> option, the messages will be sent to <code>stderr</code> . If not specified, syslog-Open Source Edition will log such messages to its internal source.                             |
| trace   | Print trace messages of how messages are processed. If syslog-Open Source Edition was started with the <code>--stderr</code> or <code>-e</code> option, the messages will be sent to <code>stderr</code> . If not specified, syslog-Open Source Edition will log such messages to its internal source. |
| debug   | Print debug messages. If syslog-Open Source Edition was started with the <code>--stderr</code> or <code>-e</code> option, the messages will be sent to <code>stderr</code> . If not specified, syslog-Open Source Edition will log such messages to its internal source.                               |

Example:



```
syslog-ng-ctl verbose --set=on
```

## The stats command

stats [options]

Use the validate command to validate the signatures and timestamps of a logstore file. The validate command has the following options:

--control=<socket> or -c	Specify the socket to use to access syslog-ng. Only needed when using a non-standard socket.
--------------------------	----------------------------------------------------------------------------------------------

Example:

```
syslog-ng-ctl stats
```

An example output:

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

## Files

/opt/syslog-ng/sbin/syslog-ng-ctl



## See also

*The syslog-*ng* Administrator Guide*

*syslog-*ng*.conf(5)*

*syslog-*ng*(8)*

If you experience any problems or need help with syslog-*ng*, visit the *syslog-*ng* mailing list*

For news and notifications about the documentation of syslog-*ng*, visit the *BalaBit Documentation Blog*.

## Author

This manual page was written by the BalaBit Documentation Team <documentation@balabit.com>.

## Copyright

Copyright © 2000-2011 BalaBit IT Security Ltd. Published under the Creative Commons Attribution-Noncommercial-No Derivative Works (by-nc-nd) 3.0 license. For details, see <http://creativecommons.org/>. The latest version is always available at <http://www.balabit.com/support/documentation>.



## Appendix 2. GNU Lesser General Public License

This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.

Copyright © 1991, 1999 Free Software Foundation, Inc.

Free Software Foundation, Inc.  
51 Franklin Street, Fifth Floor,  
Boston,  
MA  
02110-1301  
USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Version 2.1, February 1999

### 2.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method:

1. we copyright the library, and
2. we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.



To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

## 2.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

### 2.2.1. Section 0

This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".



A “library” means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The “Library”, below, refers to any such software library or work which has been distributed under these terms. A “work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term “modification”).

“Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

### 2.2.2. Section 1

You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

### 2.2.3. Section 2

You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a. The modified work must itself be a software library.
- b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful. (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)



These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

#### 2.2.4. Section 3

You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

#### 2.2.5. Section 4

You may copy and distribute the Library (or a portion or derivative of it, under [Section 2](#)) in object code or executable form under the terms of [Sections 1](#) and [2](#) above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of [Sections 1](#) and [2](#) above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

#### 2.2.6. Section 5

A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a “work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a “work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library”. The executable is therefore covered by this License. [Section 6](#) states terms for distribution of such executables.



When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under [Section 6](#).)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of [Section 6](#). Any executables containing that work also fall under [Section 6](#), whether or not they are linked directly with the Library itself.

### 2.2.7. Section 6

As an exception to the Sections above, you may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under [Sections 1](#) and [2](#) above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in [Subsection 6a](#), above, for a charge no more than the cost of performing this distribution.
- d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the “work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components



(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

### 2.2.8. Section 7

You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

### 2.2.9. Section 8

You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

### 2.2.10. Section 9

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

### 2.2.11. Section 10

Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

### 2.2.12. Section 11

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-



free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

### **2.2.13. Section 12**

If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

### **2.2.14. Section 13**

The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

### **2.2.15. Section 14**

If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

### **2.2.16. NO WARRANTY Section 15**

BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND



FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

### 2.2.17. Section 16

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

## 2.3. How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice



That's all there is to it!



## Appendix 3. GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

Free Software Foundation, Inc.  
51 Franklin Street, Fifth Floor,  
Boston, MA 02110-1301  
USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Version 2, June 1991

### 3.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.



Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## 3.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

### 3.2.1. Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

### 3.2.2. Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

### 3.2.3. Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: If the Program itself is interactive but does not



normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

#### 3.2.4. Section 3

You may copy and distribute the Program (or a work based on it, under *Section 2* in object code or executable form under the terms of *Sections 1* and *2* above provided that you also do one of the following:

- a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

#### 3.2.5. Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.



### 3.2.6. Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

### 3.2.7. Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

### 3.2.8. Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

### 3.2.9. Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.



### 3.2.10. Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

### 3.2.11. Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

### 3.2.12. NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

### 3.2.13. Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

## 3.3. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.



To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type “show w”. This is free software, and you are welcome to redistribute it under certain conditions; type “show c” for details.

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than “show w” and “show c”; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program “Gnomovision” (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.



## Appendix 4. Creative Commons Attribution Non-commercial No Derivatives (by-nc-nd) License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED. BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HEREIN IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

### 1. Definitions

- a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
- b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.
- c. "Distribute" means to make available to the public the original and copies of the Work through sale or other transfer of ownership.
- d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
- e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
- f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing;



a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

- g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
  - h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
  - i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.
2. *Fair Dealing Rights*. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.
  3. *License Grant*. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:
    - a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,
    - b. to Distribute and Publicly Perform the Work including as incorporated in Collections.The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).
  4. *Restrictions*. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
    - a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every



copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested.

- b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
- d. For the avoidance of doubt:
  - i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
  - ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,
  - iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.



cense that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

- e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.
5. *Representations, Warranties and Disclaimer* UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.
6. *Limitation on Liability*. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
7. *Termination*
  - a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
  - b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.
8. *Miscellaneous*
  - a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
  - b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
  - c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.



- d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
- e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.



# Glossary

alias IP	An additional IP address assigned to an interface that already has an IP address. The normal and alias IP addresses both refer to the same physical interface.
authentication	The process of verifying the authenticity of a user or client before allowing access to a network system or service.
auditing policy	The auditing policy determines which events are logged on host running Microsoft Windows operating systems.
BSD-syslog protocol	The old syslog protocol standard described in <a href="#">RFC 3164</a> . Sometimes also referred to as the legacy-syslog protocol.
CA	A Certificate Authority (CA) is an institute that issues certificates.
certificate	A certificate is a file that uniquely identifies its owner. Certificates contains information identifying the owner of the certificate, a public key itself, the expiration date of the certificate, the name of the CA that signed the certificate, and some other data.
client mode	In client mode, syslog-ng collects the local logs generated by the host and forwards them through a network connection to the central syslog-ng server or to a relay.
destination	A named collection of configured destination drivers.
destination driver	A communication method used to send log messages.
destination, network	A destination that sends log messages to a remote host (that is, a syslog-ng relay or server) using a network connection.
destination, local	A destination that transfers log messages within the host, for example writes them to a file, or passes them to a log analyzing application.
disk buffer	The Premium Edition of syslog-ng can store messages on the local hard disk if the central log server or the network connection to the server becomes unavailable.
disk queue	See <i>disk buffer</i> .
domain name	The name of a network, for example: <code>balabit.com</code> .
embedded log statement	A log statement that is included in another log statement to create a complex log path.
filter	An expression to select messages.



gateway	A device that connects two or more parts of the network, for example: your local intranet and the external network (the Internet). Gateways act as entrances into other networks.
high availability	High availability uses a second syslog- <i>ng</i> server unit to ensure that the logs are received even if the first unit breaks down.
host	A computer connected to the network.
hostname	A name that identifies a host on the network.
IETF-syslog protocol	The syslog-protocol standard developed by the Internet Engineering Task Force (IETF), described in <a href="#">RFC 5424-5428</a> .
key pair	A private key and its related public key. The private key is known only to the owner; the public key can be freely distributed. Information encrypted with the private key can only be decrypted using the public key.
license	The syslog- <i>ng</i> license determines the number of distinct hosts (clients and relays) that can connect to the syslog- <i>ng</i> server.
log path	A combination of sources, filters, parsers, rewrite rules, and destinations: syslog- <i>ng</i> examines all messages arriving to the sources of the logpath and sends the messages matching all filters to the defined destinations.
logstore	A binary logfile format that can encrypt, compress, and timestamp log messages.
LSH	See <i>log source host</i> .
log source host	A host or network device (including syslog- <i>ng</i> clients and relays) that sends logs to the syslog- <i>ng</i> server. Log source hosts can be servers, routers, desktop computers, or other devices capable of sending syslog messages or running syslog- <i>ng</i> .
log statement	See <i>log path</i> .
name server	A network computer storing the IP addresses corresponding to domain names.
Oracle Instant Client	The Oracle Instant Client is a small set of libraries, which allow you to connect to an Oracle Database. A subset of the full Oracle Client, it requires minimal installation but has full functionality.
output buffer	A part of the memory of the host where syslog- <i>ng</i> stores outgoing log messages if the destination cannot accept the messages immediately.
output queue	Messages from the output queue are sent to the target syslog- <i>ng</i> server. The syslog- <i>ng</i> application puts the outgoing messages directly into the output queue, unless the output queue is full. The output queue can hold 64 messages, this is a fixed value and cannot be modified.



overflow queue	See <i>output buffer</i> .
parser	A set of rules to segment messages into named fields or columns.
ping	A command that sends a message from a host to another host over a network to test connectivity and packet loss.
port	A number ranging from 1 to 65535 that identifies the destination application of the transmitted data. For example: SSH commonly uses port 22, web servers (HTTP) use port 80, and so on.
Public-key authentication	An authentication method that uses encryption key pairs to verify the identity of a user or a client.
regular expression	A regular expression is a string that describes or matches a set of strings. The syslog-ng application supports extended regular expressions (also called POSIX modern regular expressions).
relay mode	In relay mode, syslog-ng receives logs through the network from syslog-ng clients and forwards them to the central syslog-ng server using a network connection.
rewrite rule	A set of rules to modify selected elements of a log message.
template	A user-defined structure that can be used to restructure log messages or automatically generate file names.
server mode	In server mode, syslog-ng acts as a central log-collecting server. It receives messages from syslog-ng clients and relays over the network, and stores them locally in files, or passes them to other applications, for example, log analyzers.
source	A named collection of configured source drivers.
source, network	A source that receives log messages from a remote host using a network connection. The following sources are network sources: <code>tcp()</code> , <code>tcp6()</code> , <code>udp()</code> , <code>udp6()</code> .
source, local	A source that receives log messages from within the host, for example, from a file.
source driver	A communication method used to receive log messages.
SSL	See <i>TLS</i> .
syslog-ng	The syslog-ng application is a flexible and highly scalable system logging application, typically used to manage log messages and implement centralized logging.
syslog-ng agent	The syslog-ng agent for Windows is a log collector and forwarder application for the Microsoft Windows platform. It collects the log messages of the Windows-based host and forwards them to a syslog-ng server using regular or SSL-encrypted TCP connections.



syslog- <i>ng</i> client	A host running syslog- <i>ng</i> in client mode.
syslog- <i>ng</i> Premium Edition	The syslog- <i>ng</i> Premium Edition is the commercial version of the open-source application. It offers additional features, like encrypted message transfer and an agent for Microsoft Windows platforms.
syslog- <i>ng</i> relay	A host running syslog- <i>ng</i> in relay mode.
syslog- <i>ng</i> server	A host running syslog- <i>ng</i> in server mode.
TLS	Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols which provide secure communications on the Internet. The syslog- <i>ng</i> application can encrypt the communication between the clients and the server using TLS to prevent unauthorized access to sensitive log messages.
traceroute	A command that shows all routing steps (the path of a message) between two hosts.
unix domain socket	A Unix domain socket (UDS) or IPC socket (inter-procedure call socket) is a virtual socket, used for inter-process communication.



# List of syslog-ng OSE parameters

## Symbols

\$SDATA.SDID.SDNAME, 158  
\$BSDTAG, 155  
\$DATE, \$R\_DATE, \$S\_DATE, 156  
\$DAY, \$R\_DAY, \$S\_DAY, 156  
\$FACILITY, 156  
\$FACILITY\_NUM, 156  
\$FULLDATE, \$R\_FULLDATE, \$S\_FULLDATE, 156  
\$FULLHOST, 156  
\$FULLHOST\_FROM, 156  
\$HOST, 156  
\$HOST\_FROM, 157  
\$HOUR, \$R\_HOUR, \$S\_HOUR, 156  
\$ISODATE, \$R\_ISODATE, \$S\_ISODATE, 157  
\$LEVEL, 158  
\$LEVEL\_NUM, 157  
\$MIN, \$R\_MIN, \$S\_MIN, 157  
\$MONTH, \$R\_MONTH, \$S\_MONTH, 157  
\$MONTH\_ABBREV, \$R\_MONTH\_ABBREV,  
\$S\_MONTH\_ABBREV, 157  
\$MONTH\_NAME, \$R\_MONTH\_NAME,  
\$S\_MONTH\_NAME, 157  
\$MONTH\_WEEK, \$R\_MONTH\_WEEK,  
\$S\_MONTH\_WEEK, 157  
\$MSG or \$MESSAGE, 157  
\$MSGHDR, 157  
\$MSGONLY, 157  
\$PID, 158  
\$PRI, 158  
\$PRIORITY, 158  
\$PROGRAM, 158  
\$SDATA, 158  
\$SEC, \$R\_SEC, \$S\_SEC, 158  
\$SEQNUM, 158  
\$SOURCEIP, 158  
\$STAMP, \$R\_STAMP, \$S\_STAMP, 159  
\$TAG, 159  
\$TAGS, 159  
\$TZ, \$R\_TZ, \$S\_TZ, 159  
\$TZOFFSET, \$R\_TZOFFSET, \$S\_TZOFFSET, 159  
\$UNIXTIME, \$R\_UNIXTIME, \$S\_UNIXTIME, 159  
\$WEEK, \$R\_WEEK, \$S\_WEEK, 159  
\$WEEKDAY, \$R\_WEEKDAY, \$S\_WEEKDAY, 159  
\$WEEK\_ABBREV, \$R\_WEEK\_ABBREV,  
\$S\_WEEK\_ABBREV, 159  
\$WEEK\_DAY, \$R\_WEEK\_DAY, \$S\_WEEK\_DAY, 159  
\$WEEK\_DAY\_NAME, \$R\_WEEK\_DAY\_NAME,  
\$S\_WEEK\_DAY\_NAME, 160  
\$YEAR, \$R\_YEAR, \$S\_YEAR, 159  
.SDATA.SDID.SDNAME, 158  
@define, 41  
@distance, 180  
@include, 41

## A

action, 188  
actions, 180, 188  
AND, 129

## B

bad\_hostname(), 137  
block, 42  
block arguments, 43  
BSDTAG, 155

## C

catchall,  
ca\_dir(), 151  
cert\_file(), 151  
chain\_hostnames(), 137  
check\_hostname(), 137  
class, 186  
columns(), 102  
condition, 188  
condition(), 163  
context-id, 186  
context-scope, 186  
context-timeout, 186  
create\_dirs(), 87, 138  
crl\_dir(), 151  
csv-parser, 170  
Custom macros, 155

## D

database(), 102  
DATE, R\_DATE, S\_DATE, 156  
DAY, R\_DAY, S\_DAY, 156



default-facility(), 48  
default-priority(), 48  
delimiters, 170  
description — pattern, 187  
description — ruleset, 184  
dir\_group(), 87, 138  
dir\_owner(), 87, 138  
dir\_perm(), 87, 138  
dns\_cache(), 138  
dns\_cache\_expire(), 138  
dns\_cache\_expire\_failed(), 139  
dns\_cache\_hosts(), 139  
dns\_cache\_size(), 139  
dont-create-tables, 103  
door(), 61  
drop-invalid, 170

## E

echo, 160  
encoding(), 49, 74, 80  
eq, 130  
escape-backslash, 171  
escape-double-char, 171  
escape-none, 171  
example, 187  
examples, 187  
explicit-commits, 103

## F

FACILITY, 156  
facility(), 133-134  
FACILITY\_NUM, 156  
fallback,  
file, 56  
file(), 49  
filter(), 134  
final,  
flags, 121, 124  
    empty-lines, 49, 52, 57, 61, 65, 74, 80  
    kernel, 49, 52, 57, 61, 65, 74, 80  
    no-hostname, 49, 52, 57, 61, 65, 74, 80  
    no-multi-line, 49, 52, 57, 61, 65, 74, 80  
    no-parse, 49, 52, 57, 61, 65, 74, 80  
    no\_multi\_line, 88, 92, 96, 107, 112, 117  
    store-legacy-msghdr, 49, 52, 57, 61, 65, 74, 80  
    syslog-protocol, 88, 92, 96, 107, 112, 117  
    validate-utf8, 49, 52, 57, 61, 65, 74, 80

flags(), 49, 52, 57, 61, 65, 74, 80, 88, 92, 96, 107, 112, 117, 170  
    for SQL destinations, 102  
flow-control, 125, 127  
flush\_lines(), 88, 92, 96, 103, 108, 113, 117, 139  
flush\_timeout(), 88, 92, 96, 103, 108, 113, 118, 139  
follow\_freq(), 50, 53, 57-58, 62, 66  
frac\_digits(), 88, 93, 96, 103, 108, 113, 118  
fsync(), 88  
FULLDATE, R\_FULLDATE, S\_FULLDATE, 156  
FULLHOST, 156  
FULLHOST\_FROM, 156

## G

ge, 130  
glob, 166  
global, 165-166  
global — context-scope, 186  
greedy, 171  
grep, 160  
group(), 81, 89, 93, 140  
gt, 130

## H

HOST, 156  
host — context-scope, 186  
host(), 103, 134  
HOST\_FROM, 157  
host\_override(), 66, 75, 81  
HOUR, R\_HOUR, S\_HOUR, 156

## I

id — rule, 186  
id — ruleset, 184  
if, 161  
ignore-case, 165-166  
indexes(), 104  
ip() or localip(), 66, 75  
ipv4-to-int, 161  
ip\_tos(), 66, 108, 113  
ip\_ttl(), 67, 108, 113  
ISODATE, R\_ISODATE, S\_ISODATE, 157

## K

keep-alive(), 67, 75, 81, 109, 114, 118  
keep\_hostname(), 67, 76, 140  
keep\_timestamp(), 50, 54, 58, 62, 68, 76, 82, 140



key\_file(), 152

## L

le, 130  
LEVEL, 158  
level() or priority(), 134  
LEVEL\_NUM, 157  
localip(), 109, 114  
localport(), 109, 114  
local\_time\_zone(), 89, 104  
log\_fetch\_limit(), 50, 54, 59, 62, 68, 76, 82  
log\_fifo\_size(), 89, 93, 97, 104, 109, 118, 141  
log\_iw\_size(), 50, 54, 59, 63, 68, 77, 82  
log\_msg\_size(), 50, 54, 59, 63, 68, 77, 82, 141  
log\_prefix() (DEPRECATED), 51, 54, 59, 63, 68, 77, 82  
lt, 130

## M

mark(), 141  
mark\_freq(), 141  
match, 188  
match(), 134  
max-connections(), 68, 77, 82  
message, 189  
message(), 135  
MIN, R\_MIN, S\_MIN, 157  
MONTH, R\_MONTH, S\_MONTH, 157  
MONTH\_ABBREV, R\_MONTH\_ABBREV,  
S\_MONTH\_ABBREV, 157  
MONTH\_NAME, R\_MONTH\_NAME,  
S\_MONTH\_NAME, 157  
MONTH\_WEEK, R\_MONTH\_WEEK,  
S\_MONTH\_WEEK, 157  
MSG or MESSAGE, 157  
MSGHDR, 157  
MSGONLY, 157

## N

name — pattern value, 187  
name — ruleset, 184  
name — test\_value, 188  
name — value — action, 189  
ne, 130  
netmask(), 135  
nobackref, 166  
normalize\_hostnames(), 141  
NOT, 129

null(), 104

## O

optional(), 51, 54, 59, 63, 83  
OR, 129  
overwrite\_if\_older(), 89  
owner(), 83, 89, 93, 141

## P

pacct(), 56  
pad\_size(), 51, 55, 59, 63, 69, 77, 83, 90, 93  
password(), 105  
pattern — rule, 187  
pattern — ruleset, 185  
patterndb, 184  
patterns, 185  
patterns — rule, 187  
pcre, 165  
peer\_verify(), 152  
perm(), 83, 90, 94, 142  
PID, 158  
pipe(), 55  
port(), 105  
port() or destport(), 109, 114  
port() or localport(), 69, 77  
posix, 165  
PRI, 158  
PRIORITY, 158  
process — context-scope, 186  
program, 60  
PROGRAM, 158  
program — context-scope, 186  
program(), 135  
program\_override(), 51, 55, 60, 64, 69, 78, 83  
provider, 186  
pubdate, 184

## Q

quote-pairs(), 171

## R

rate, 188  
recv\_time\_zone(), 142  
rewrite(), 162  
root, 43  
rule, 185  
rules, 185



ruleset, 184

## S

SDATA, 158  
SEC, R\_SEC, S\_SEC, 158  
send\_time\_zone(), 142  
SEQNUM, 158  
set(), 163  
source(), 135  
SOURCEIP, 158  
so\_broadcast(), 69, 109, 114, 118  
so\_keepalive(), 69, 78, 84, 110, 115, 118  
so\_rcvbuf(), 69, 78, 84, 110, 115, 119  
so\_sndbuf(), 70, 110, 115, 119  
spoof\_source(), 110, 115  
STAMP, R\_STAMP, S\_STAMP, 159  
stats\_freq(), 142  
stats\_level(), 142  
store-matches, 165-166  
string, 166  
strip-whitespace, 171  
subst(), 162  
suppress(), 90, 94, 97, 110, 115, 119  
sync() or sync\_freq() (DEPRECATED), 143  
system(), 71

## T

table(), 105  
TAG, 159  
tag — rule, 189  
TAGS, 159  
tags — rule, 189  
tags(), 51, 55, 60, 64, 70, 78, 84, 135, 177  
tcp-keep-alive(), 70, 78  
template(), 90, 94, 97, 110, 115, 119, 171  
template\_escape(), 90, 94, 97, 111, 116, 119  
test\_message, 188  
test\_value, 188  
test\_values, 188  
throttle(), 91, 94, 97, 111, 116, 120  
timeout, 188  
time\_reap(), 91, 143  
time\_reopen(), 143  
time\_sleep(), 143  
time\_zone(), 52, 55, 60, 64, 70, 79, 84, 91, 95, 98, 105,  
111, 116, 120, 139  
tls(), 71, 79, 111, 116  
transport, 111

transport(), 70  
trigger, 188  
trusted\_dn(), 152  
trusted\_keys(), 152  
ts\_format(), 91, 95, 98, 112, 116, 120, 143  
type(), 106, 164  
TZ, R\_TZ, S\_TZ, 159  
TZOFFSET, R\_TZOFFSET, S\_TZOFFSET, 159

## U

unicode, 166  
UNIXTIME, R\_UNIXTIME, S\_UNIXTIME, 159  
url — pattern, 187  
url — ruleset, 185  
urls — pattern, 187  
username(), 106  
use\_dns(), 67, 71, 76, 79, 140, 143  
use\_fqdn(), 71, 79, 144  
use\_time\_recv() (DEPRECATED), 144  
utf8, 165-166

## V

value — action, 189  
value — pattern, 187  
value(), 135  
values — action, 189  
values — pattern, 187  
values(), 106  
version, 184

## W

WEEK, R\_WEEK, S\_WEEK, 159  
WEEKDAY, R\_WEEKDAY, S\_WEEKDAY, 159  
WEEK\_ABBREV, R\_WEEK\_ABBREV,  
S\_WEEK\_ABBREV, 159  
WEEK\_DAY, R\_WEEK\_DAY, S\_WEEK\_DAY, 159  
WEEK\_DAY\_NAME, R\_WEEK\_DAY\_NAME,  
S\_WEEK\_DAY\_NAME, 160

## Y

YEAR, R\_YEAR, S\_YEAR, 159



# Index

## Symbols

\$SDATA.SDID.SDNAME, 158  
\$BSDTAG, 155  
\$DATE, \$R\_DATE, \$S\_DATE, 156  
\$DAY, \$R\_DAY, \$S\_DAY, 156  
\$FACILITY, 156  
\$FACILITY\_NUM, 156  
\$FULLDATE, \$R\_FULLDATE, \$S\_FULLDATE, 156  
\$FULLHOST, 156  
\$FULLHOST\_FROM, 156  
\$HOST, 156  
\$HOST\_FROM, 157  
\$HOUR, \$R\_HOUR, \$S\_HOUR, 156  
\$ISODATE, \$R\_ISODATE, \$S\_ISODATE, 157  
\$LEVEL, 158  
\$LEVEL\_NUM, 157  
\$MIN, \$R\_MIN, \$S\_MIN, 157  
\$MONTH, \$R\_MONTH, \$S\_MONTH, 157  
\$MONTH\_ABBREV, \$R\_MONTH\_ABBREV, \$S\_MONTH\_ABBREV, 157  
\$MONTH\_NAME, \$R\_MONTH\_NAME, \$S\_MONTH\_NAME, 157  
\$MONTH\_WEEK, \$R\_MONTH\_WEEK, \$S\_MONTH\_WEEK, 157  
\$MSG or \$MESSAGE, 157  
\$MSGHDR, 157  
\$MSGONLY, 157  
\$PID, 158  
\$PRI, 158  
\$PRIORITY, 158  
\$PROGRAM, 158  
\$SDATA, 158  
\$SEC, \$R\_SEC, \$S\_SEC, 158  
\$SEQNUM, 158  
\$SOURCEIP, 158  
\$STAMP, \$R\_STAMP, \$S\_STAMP, 159  
\$TAG, 159  
\$TAGS, 159  
\$TZ, \$R\_TZ, \$S\_TZ, 159  
\$TZOFFSET, \$R\_TZOFFSET, \$S\_TZOFFSET, 159  
\$UNIXTIME, \$R\_UNIXTIME, \$S\_UNIXTIME, 159  
\$WEEK, \$R\_WEEK, \$S\_WEEK, 159  
\$WEEKDAY, \$R\_WEEKDAY, \$S\_WEEKDAY, 159

\$WEEK\_ABBREV, \$R\_WEEK\_ABBREV,  
\$S\_WEEK\_ABBREV, 159  
\$WEEK\_DAY, \$R\_WEEK\_DAY, \$S\_WEEK\_DAY, 159  
\$WEEK\_DAY\_NAME, \$R\_WEEK\_DAY\_NAME, \$S\_WEEK\_DAY\_NAME, 160  
\$YEAR, \$R\_YEAR, \$S\_YEAR, 159  
.SDATA.SDID.SDNAME, 158  
@define, 41  
@distance, 180  
@include, 41

## A

action, 188  
actions, 180, 188  
AIX  
    installing syslog-ng, 22  
    redirecting errorlog to syslog-ng, 23  
alerting, 180  
AND, 129  
artificial ignorance  
    message classification, 182  
authentication, 145-146

## B

bad\_hostname(), 137  
batch processing, 196  
block, 42  
block arguments, 43  
boolean operators, 129  
BSDTAG, 155

## C

catchall,  
ca\_dir(), 151  
CentOS  
    installing syslog-ng, 22  
certificates, 145  
cert\_file(), 151  
chain\_hostnames(), 137  
check\_hostname(), 137  
chroots, 198  
class, 186  
classifying messages  
    concepts of, 173  
configuration, 176  
creating databases, 184



filtering, 177  
pattern matching concepts, 175

client mode, 6

columns(), 102  
comparing values, 130  
compatibility with Snare, 49, 53, 58, 62, 66, 75, 81  
compiling syslog-ng OSE, 23

condition, 188  
condition(), 163  
conditional rewrites, 163

configuration file  
    default configuration, 32-33  
    including other files, 41

configuration snippets, 42

context of messages, 179

context-id, 186

context-scope, 186

context-timeout, 186

Coordinated Universal Time, 9

core files, 194

correlating messages, 179

create\_dirs(), 87, 138

crl\_dir(), 151

CSV parsers, 170

csv-parser, 170

Custom macros, 155

**D**

database(), 102  
DATE, R\_DATE, S\_DATE, 156  
DAY, R\_DAY, S\_DAY, 156  
daylight saving changes, 8  
default-facility(), 48  
default-priority(), 48  
deleting syslog-ng, 25  
delimiters, 170  
description — pattern, 187  
description — ruleset, 184  
destination drivers, 8, **85**  
    database driver, 98, 102  
    file() driver, 86-87  
    list of, 86, 207  
    pipe() driver, 91-92  
    program() driver, 95-96  
    sql() driver, 98, 102  
    syslog() driver, 106-107  
    tcp() driver, 112  
    tcp6() driver, 112

    udp() driver, 112  
    udp6() driver, 112  
    unix-dgram() driver, 117  
    unix-stream() driver, 117  
    userpty() driver, 120

destinations, 4, 8, **85**  
    defining, 45, 85  
    FreeTDS configuration, 26  
    Microsoft SQL Server configuration, 26  
    MSSQL configuration, 26  
    sql() configuration, 99-100, 105

dir\_group(), 87, 138  
dir\_owner(), 87, 138  
dir\_perm(), 87, 138  
discarding messages, 136

disk queue  
    disk buffer, 127

dns\_cache(), 138  
dns\_cache\_expire(), 138  
dns\_cache\_expire\_failed(), 139  
dns\_cache\_hosts(), 139  
dns\_cache\_size(), 139  
dont-create-tables, 103

door(), 61

download  
    pattern databases, 178

drop-invalid, 170

dropping messages, 136

**E**

echo, 160  
embedded log statements, 122  
encoding(), 49, 74, 80  
encrypting log messages, 145-146  
environmental variables, 41  
eq, 130  
error solving, 193  
escape-backslash, 171  
escape-double-char, 171  
escape-none, 171  
escaping special characters, 164  
example, 187  
examples, 187  
explicit-commits, 103

**F**

facilities, 12, 14, 134, 196  
FACILITY, 156



facility(), 133-134  
FACILITY\_NUM, 156  
fail-over, 10  
failure script, 195  
fallback,  
fd limit, 87  
feature releases, 9  
file, 56  
file descriptors, 87  
file(), 49  
filter functions  
    list of, 133, 208  
filter(), 134  
filtering  
    .classifier\_class, 177  
    on message class, 177  
filtering rewrites, 163  
filters, 4, 8, **129**, 166, 197  
    AND, OR, NOT, 129  
    boolean operators, 129  
    comparing values, 130  
    control characters, 132  
    defining, 129  
    facilities, , 133  
    facility and priority (level) ranges, 134  
    priorities, 134  
    reference, 132  
    tags, 132  
    wildcards, 131  
final,  
flags, 121, 124  
    empty-lines, 49, 52, 57, 61, 65, 74, 80  
    kernel, 49, 52, 57, 61, 65, 74, 80  
    no-hostname, 49, 52, 57, 61, 65, 74, 80  
    no-multi-line, 49, 52, 57, 61, 65, 74, 80  
    no-parse, 49, 52, 57, 61, 65, 74, 80  
    no\_multi\_line, 88, 92, 96, 107, 112, 117  
    store-legacy-msghdr, 49, 52, 57, 61, 65, 74, 80  
    syslog-protocol, 88, 92, 96, 107, 112, 117  
    validate-utf8, 49, 52, 57, 61, 65, 74, 80  
flags(), 49, 52, 57, 61, 65, 74, 80, 88, 92, 96, 107, 112, 117, 170  
    for SQL destinations, 102  
flow-control, 125, 127  
    example, 128  
    multiple destinations, 127  
flush\_lines(), 88, 92, 96, 103, 108, 113, 117, 139  
flush\_timeout(), 88, 92, 96, 103, 108, 113, 118, 139  
follow\_freq(), 50, 53, 57-58, 62, 66  
formatting messages, 153  
frac\_digits(), 88, 93, 96, 103, 108, 113, 118  
fsync(), 88  
FULLDATE, R\_FULLDATE, S\_FULLDATE, 156  
FULLHOST, 156  
FULLHOST\_FROM, 156

## G

ge, 130  
generating alerts, 180  
glob, 166  
glob patterns, 166  
global, 165-166  
global objects, 7  
global options, **137**  
    reference, 137  
global variables, 41  
global — context-scope, 186  
greedy, 171  
grep, 160  
group(), 81, 89, 93, 140  
gt, 130

## H

HOST, 156  
host — context-scope, 186  
host(), 103, 134  
HOST\_FROM, 157  
host\_override(), 66, 75, 81  
HOUR, R\_HOUR, S\_HOUR, 156

## I

id — rule, 186  
id — ruleset, 184  
if, 161  
ignore-case, 165-166  
indexes(), 104  
installation path, 16  
installing syslog-ng, **16**  
    from DEB package, 23  
    from RPM package, 22  
    in silent mode, 21  
    on AIX, 22  
    on CentOS, 22  
    on clients and relays, 17  
    on logservers, 19



on Red Hat Enterprise Server, 22  
on SUSE Linux Enterprise Server, 22  
installing syslog-ng OSE from source, 23  
`ip()` or `localip()`, 66, 75  
`ipv4-to-int`, 161  
`ip_tos()`, 66, 108, 113  
`ip_ttl()`, 67, 108, 113  
`ISODATE`, `R_ISODATE`, `S_ISODATE`, 157

## K

`keep-alive()`, 67, 75, 81, 109, 114, 118  
`keep_hostname()`, 67, 76, 140  
`keep_timestamp()`, 50, 54, 58, 62, 68, 76, 82, 140  
`key_file()`, 152

## L

`le`, 130  
`LEVEL`, 158  
`level()` or `priority()`, 134  
`LEVEL_NUM`, 157  
local time, 12, 14  
`localip()`, 109, 114  
`localport()`, 109, 114  
`local_time_zone()`, 89, 104  
log messages, structure, 10

- BSD-syslog protocol, 10
- IETF-syslog protocol, 12
- legacy-syslog protocol, 10
  - RFC 3164, 10
  - RFC 5424, 12

log paths, 4, **121**

- defining, 121
- flags, 121, 124
- flow-control, 125, 127-128

log pipes

- embedded log statements, 122

log statements, 8

- embedded, **122**
- log paths, 4

log statistics, 191

- on unix-socket, 191

logging procedure, 4

`log_fetch_limit()`, 50, 54, 59, 62, 68, 76, 82

`log_fifo_size()`, 89, 93, 97, 104, 109, 118, 141

`log_iw_size()`, 50, 54, 59, 63, 68, 77, 82

`log_msg_size()`, 50, 54, 59, 63, 68, 77, 82, 141

`log_prefix()` (DEPRECATED), 51, 54, 59, 63, 68, 77, 82

losing messages, 193

`lt`, 130

## M

`macros`, 8, 153

- default value, 154
- in filenames, 155
- patterndb tags, 159
- reference, 155
- SDATA, 158

`mark()`, 141

`mark_freq()`, 141

`match`, 188

`match()`, 134

`max-connections()`, 68, 77, 82

`message`, 189

- statistics, 191

`message classification`, 176-177, 184

`message context`, 179

`message correlation`, 179

`message counters`, 191

`message facilities`, 12, 14, 134

`message filtering`

- using parsers, 177

`message loss`, 193

`message parsing`, 168, 176-177

`message statistics`, 191

`message templates`, 153

`message triggers`, 180

`message()`, 135

`Microsoft SQL`

- `sql()` configuration, 100

`Microsoft SQL Server configuration`, 26

`MIN`, `R_MIN`, `S_MIN`, 157

modes of operation, 6

- client mode, 6
- relay mode, 7
- server mode, 7

`MONTH`, `R_MONTH`, `S_MONTH`, 157

`MONTH_ABBREV`, `R_MONTH_ABBREV`, `S_MONTH_ABBREV`, 157

`MONTH_NAME`, `R_MONTH_NAME`, `S_MONTH_NAME`, 157

`MONTH_WEEK`, `R_MONTH_WEEK`, `S_MONTH_WEEK`, 157

`MSG` or `MESSAGE`, 157

`MSGHDR`, 157

`MSGONLY`, 157

`MSSQL`



sql() configuration, 100  
mutual authentication, 145, 148

## N

name resolution, 196-197  
  local, 198  
name — pattern value, 187  
name — ruleset, 184  
name — test\_value, 188  
name — value — action, 189  
ne, 130  
netmask(), 135  
nobackref, 166  
normalize\_hostnames(), 141  
NOT, 129  
null(), 104  
number of open files, 87

## O

optimizing regular expressions, 166  
optimizing syslog-ng performance, 197  
  regular expressions, 166  
optional(), 51, 54, 59, 63, 83  
options, 8  
  reference, 137  
OR, 129  
Oracle  
  sql() configuration, 99  
output buffer, 125, 127  
output queue, 127  
overflow queue  
  output buffer, 127  
overriding facility, 45  
overwrite\_if\_older(), 89  
owner(), 83, 89, 93, 141

## P

pacct(), 56  
pad\_size(), 51, 55, 59, 63, 69, 77, 83, 90, 93  
parallel connections, 196  
parameters  
  log\_fetch\_limit(), 125, 127, 196  
  log\_fifo\_size(), 125, 127, 196  
  log\_iw\_size(), 126-127  
  max\_connections(), 126-127, 196  
  time\_sleep(), 196  
parsers, 4, 8, 168, 176-177

parsing messages, 168, 176-177, 182  
  concepts of, 168  
  filtering parsed messages, 177

password(), 105  
pattern database, 176-177, 184, 189

  creating parsers, 182  
  structure of, 174  
  using the results, 177

pattern database schema, 184

pattern databases  
  concepts of, 173  
  correlating messages, 179  
  pattern matching precedence, 175

pattern matching

  procedure of, 175

pattern — rule, 187

pattern — ruleset, 185

patternedb, 184

  download, 178

patterns, 185

patterns — rule, 187

pcre, 165

peer\_verify(), 152

perm(), 83, 90, 94, 142

PID, 158

pipe(), 55

port(), 105

port() or destport(), 109, 114

port() or localport(), 69, 77

posix, 165

PostgreSQL

  sql() configuration, 99

preventing message loss

  flow-control, 125, 127

PRI, 158

PRIORITY, 158

process accounting, 56

process — context-scope, 186

program, 60

PROGRAM, 158

program — context-scope, 186

program(), 135

program\_override(), 51, 55, 60, 64, 69, 78, 83

provider, 186

pubdate, 184

## Q

quote-pairs(), 171



## R

rate, 188  
reading messages from external applications, 57  
`recv_time_zone()`, 142  
Red Hat Enterprise Server  
    installing syslog-ng, 22  
regular expressions, **129, 164, 166, 197**  
    case-insensitive, 164  
    escaping, 164  
    pcre, 165  
    posix, 131  
relay mode, 7  
releases, 9  
removing syslog-ng, 25  
replacing message text, 162  
reusing snippets, 42  
rewrite if, 163  
rewrite rules, 4, 8, 162  
`rewrite()`, 162  
rewriting messages, 162  
    concepts of, 162  
    conditional rewrites, 163  
root, 43  
rule, 185  
rules, 185  
ruleset, 184

## S

scl  
    `system()`, 71  
SDATA, 158  
SEC, R\_SEC, S\_SEC, 158  
secondary messages, 180  
sedding messages, 162  
segmenting messages, 168, 170  
`send_time_zone()`, 142  
SEQNUM, 158  
server mode, 7  
`set()`, 163  
setting facility, 45  
setting message fields, 163  
skipping messages, 136  
Snare  
    receiving Snare-compatible messages, 49, 53, 58, 62, 66, 75, 81  
Snare-compatibility, 49, 53, 58, 62, 66, 75, 81  
source drivers, 7, **45**

`file()` driver, 47-48  
list of, 47, 207  
`pacct()` driver, 56  
`pipe()` driver, 52  
`program()` driver, 57  
reference, 45  
`sun-streams()` driver, 60-61  
`syslog()` driver, 64-65  
`system()` driver, 71  
`tcp()` driver, 72, 74  
`tcp6()` driver, 72, 74  
`udp()` driver, 72, 74  
`udp6()` driver, 72, 74  
`unix-dgram()` driver, 80  
`unix-stream()` driver, 80  
`source()`, 135  
SOURCEIP, 158  
sources, 4, 7, **45**  
    on different platforms, 46  
`so_broadcast()`, 69, 109, 114, 118  
`so_keepalive()`, 69, 78, 84, 110, 115, 118  
`so_rcvbuf()`, 69, 78, 84, 110, 115, 119  
`so_sndbuf()`, 70, 110, 115, 119  
splitting messages, 168, 170  
`spoof_source()`, 110, 115  
SQL NULL values, 105  
stable releases, 9  
STAMP, R\_STAMP, S\_STAMP, 159  
statistics, 191  
`stats_freq()`, 142  
`stats_level()`, 142  
store-matches, 165-166  
strace, 194  
string, 166  
string comparison, 130  
strip-whitespace, 171  
STRUCTURED-DATA, 158  
`subst()`, 162  
supported architectures, 3  
supported operating systems, 3  
`suppress()`, 90, 94, 97, 110, 115, 119  
SUSE Linux Enterprise Server  
    installing syslog-ng, 22  
`sync()` or `sync_freq()` (DEPRECATED), 143  
syslog-ng  
    troubleshooting, 193  
syslog-ng agent  
    Snare-compatibility, 49, 53, 58, 62, 66, 75, 81



syslog-*ng* binaries  
    location of, 16  
syslog-*ng* clients  
    configuring, 32  
syslog-*ng* relays  
    configuring, 35  
syslog-*ng* servers  
    configuring, 34  
**syslog-*ng*.conf, 38**  
    environmental variables, 41  
    global variables, 41  
    includes, 41  
system(), 71

**T**

table(), 105  
TAG, 159  
tag — rule, 189  
tagging messages, 132, 189  
tags, 132, 189  
    as macro, 159  
TAGS, 159  
tags — rule, 189  
tags(), 51, 55, 60, 64, 70, 78, 84, 135, 177  
tcp-keep-alive(), 70, 78  
template functions, 160  
    embedding, 161  
template(), 90, 94, 97, 110, 115, 119, 171  
templates, 8, 153, **154**  
    defining, 155  
    example, 155  
    template functions, 160  
template\_escape(), 90, 94, 97, 111, 116, 119  
test\_message, 188  
test\_value, 188  
test\_values, 188  
throttle(), 91, 94, 97, 111, 116, 120  
timeout, 188  
timestamp, 9, 12, 14, 196  
timezone  
    in chroots, 199  
timezones, 8-9  
time\_reap(), 91, 143  
time\_reopen(), 143  
time\_sleep(), 143  
time\_zone(), 52, 55, 60, 64, 70, 79, 84, 91, 95, 98, 105, 111, 116, 120, 139  
TLS, 64-65, 73, 145  
    configuring, 146, 148  
    reference, 151  
tls(), 71, 79, 111, 116  
transport, 111  
transport layer security  
    TLS, 145  
transport(), 70  
trigger, 188  
triggered messages, 180  
triggers, 180  
troubleshooting, 193  
    core files, 194  
    failure script, 195  
    strace, 194  
    syslog-*ng*, 194-195  
    truss, 194  
    tusc, 194  
truss, 194  
trusted\_dn(), 152  
trusted\_keys(), 152  
ts\_format(), 91, 95, 98, 112, 116, 120, 143  
tusc, 194  
type(), 106, 164  
TZ, R\_TZ, S\_TZ, 159  
TZOFFSET, R\_TZOFFSET, S\_TZOFFSET, 159

**U**

ulimit, 87  
unicode, 166  
uninstalling syslog-*ng*, 25  
UNIXTIME, R\_UNIXTIME, S\_UNIXTIME, 159  
url — pattern, 187  
url — ruleset, 185  
urls — pattern, 187  
username(), 106  
use\_dns(), 67, 71, 76, 79, 140, 143  
use\_fqdn(), 71, 79, 144  
use\_time\_recv() (DEPRECATED), 144  
UTC, 9  
utf8, 165-166

**V**

value comparison, 130  
value — action, 189  
value — pattern, 187  
value(), 135  
values — action, 189  
values — pattern, 187



values(), 106  
version, 184

## W

WEEK, R\_WEEK, S\_WEEK, 159  
WEEKDAY, R\_WEEKDAY, S\_WEEKDAY, 159  
WEEK\_ABBREV, R\_WEEK\_ABBREV,  
S\_WEEK\_ABBREV, 159  
WEEK\_DAY, R\_WEEK\_DAY, S\_WEEK\_DAY, 159  
WEEK\_DAY\_NAME, R\_WEEK\_DAY\_NAME,  
S\_WEEK\_DAY\_NAME, 160

## Y

YEAR, R\_YEAR, S\_YEAR, 159